ioProgrammo Anno XI - N°10 (119)

IL TUO CODICE SU SECOND LIFE • PROGRAMMA IL TUO ROBOT • TRACCE ISO DAI NOSTRI PROGRAMMI • EFFETTI SPECIALI NELLE APPLICAZIONI

EDIZIONI MASTER

SPECIALE MACCHINE VIRTUALI

ECCO COME RISPARMIARE SULL'HARDWARE ED AVERE SEMPRE A DISPOSIZIONE MOLTI PC SU CUI EFFETTUARE I TEST

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90

VERSIONE STANDARD RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • OTTOBRE 2007 • ANNO XI, N.10 (119)

Il tuo codice su SECOND® LIFE

Crea, sviluppa e vendi gli oggetti che popolano l'universo digitale

- ✔ **LE BASI** usa subito il Linden Scripting Language e l'ambiente di sviluppo
- ✔ **HELLO WORLD** disegna il tuo primo oggetto e fallo parlare quando lo tocchi con il mouse
- ✔ **IL MOVIMENTO** fai spostare le tue creazioni, seguendo fedelmente le leggi della fisica
- ✔ **EFFETTI COMPOSTI** metti insieme più oggetti e animali in sincronia

XPS L'ALTERNATIVA MICROSOFT AL PDF

Ecco come modificare le tue applicazioni affinché supportino il **nuovo formato!**

.NET
DAI UNA CACHE AL TUO SOFTWARE

Quando i dati sono tanti, ecco le tecniche per ottenere comunque prestazioni elevate

GRANDE FRATELLO CON IL GPS

Ricevi le coordinate e mostra la posizione dell'apparecchio su una mappa di Google

JAVASCRIPT
FRAME NASCOSTI QUASI COME AJAX

Sfrutta gli Hidden frame. Ottimizza il caricamento delle pagine ed aggira il problema del reload

CORSI BASE

- **WxWidgets** Il posizionamento dei controlli
- **PocketPC** Programmazione degli Smartphone
- **Java Server Faces** Gestione dei dati attraverso JDBC
- **UML** Cosa sono e come funzionano i diagrammi di stato

PROGRAMMA IL TUO ROBOT

Con Microsoft Studio Robotics bastano pochi click di mouse per progettare un sistema completo e simularne il comportamento prima di assemblare i pezzi

.NET
TRACCE ISO DAI NOSTRI PROGRAMMI

Leggi e crea i dati in formato CD o DVD direttamente dal tuo software

JAVA
EFFETTI SPECIALI PER LE APPLICAZIONI

Crea interfacce grafiche sofisticate con le Swing Bug

JAVASCRIPT
TRASFORMAZIONI XSLT LATO SERVER

Stessi dati, output diverso. Basta cambiare pochi parametri

SOLUZIONI ALGORITMI DI SCHEDULING: QUANDO VUOI PROGRAMMARE UNA SERIE DI ATTIVITÀ CI SONO MOLTE POSSIBILI SOLUZIONI, ECCO QUELLE PIÙ EFFICACI

Anno XI - N.ro 10 (119) - Ottobre 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, A. Galeazzi, F. Grimaldi, E. Viale, V. Arconzo, A. Pelleriti, L. Corrias, G. Malaga, F. Fortino
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI

ITALIA: Abbonamento Annuale: IOPROGRAMMO (11 NUMERI) €59,90 SCONTO 21% SUL PREZZO DI COPERTINA DI €75,90 · IOPROGRAMMO CON LIBRO (11 NUMERI) €75,90 SCONTO 30% SUL PREZZO DI COPERTINA DI €108,90 OFFERTE VALIDE FINO AL 30/11/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano
Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ **tel. 02 831212**
@ **e-mail: servizioabbonati@edmaster.it**

Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma
Finito di stampare nel mese di Settembre 2007

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

EDIZIONI MASTER
Certificato UNI EN ISO 14001 N.9191.EMAS
A.N.E.S. Associazione Nazionale Editoria Periodica Specializzata
Certificato n.6006 del 04/12/2006
ITportal L'Universo Tecnologico www.itportal.it

Questo mese su ioProgrammo

NUOVI MERCATI

Chi legge ioProgrammo da qualche anno è abituato al carico di innovazione tecnologica che abitualmente presentiamo in queste pagine, così che spesso risulta difficile fare una pausa di riflessione e riaggregare la pura tecnica in termini di spinta di mercato. Eppure in questo numero appaiono evidente tre linee guida: scripting embedded, automazione robotica, programmazione dei device mobili. Se la terza, appare ormai lontana dall'essere una novità, rimane comunque un mercato affascinante per le potenzialità ancora non completamente espresse che la miniaturizzazione delle apparecchiature di comunicazione può offrire. La mobility rappresenta senza dubbio un futuro non lontano e anche se lentamente, la convergenza fra applicazioni desktop e mobile sembra offrire numerosi sbocchi a noi programmatori. D'altra parte gli altri due temi rappresentano invece una novità assoluta. Da un lato parliamo di programmazione embedded all'interno di un mondo virtuale, dall'altro questo tipo di tecnologia può essere facilmente espansa per abbracciare tipologie di programmazione più generali. Pensate alla programmazione di plugin per i videogame ma anche allo sviluppo che un vostro software potrebbe avere se dotato di un linguaggio di programmazione interna. Qualunque vostra applicazione potrebbe ottenere un enorme vantaggio se concepita con una tecnica del genere. Allo stesso modo esistono già software conosciutissimi per i quali è possibile sviluppare plugin utilizzando un linguaggio di scripting embedded, pensate a World Of Warcraft come a Passpartout per citarne solo un paio. Il terzo tema che affrontiamo in questo numero è invece un assoluta novità. Si tratta di applicare i concetti di drag & drop dei componenti alla robotica. Si certo siamo ancora lontani dalla progettazione di un robot per l'utente comune, ma neanche poi tanto.

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

Il tuo codice su

Crea, sviluppa e vendi gli oggetti che popolano l'universo digitale

LE BASI: usa subito il Linden Scripting Language e l'ambiente di sviluppo

HELLO WORLD: disegna il tuo primo oggetto e fallo parlare quando lo tocchi con il mouse

IL MOVIMENTO: fai spostare le tue creazioni, seguendo le leggi della fisica

EFFETTI COMPOSTI: metti insieme più oggetti e animali in sincronia

CREA IL TUO ROBOT A COLPI DI CLICK

Microsoft ha appena presentato il suo "Visual Studio Robotics", il primo ambiente RAD e visuale per lo sviluppo rapido di applicazioni destinate all'automazione robotica. Ecco come installarlo e usarlo facilmente

pag. 24

IOPROGRAMMO WEB

Trasformazioni XSLT lato server . . . . **pag. 30**
XSLT consente di ottenere un documento in un qualunque formato a partire da una base XML. Ad esempio dagli stessi dati si può ottenere un file PDF oppure un file DOC. Impariamo come integrare questa tecnica con i linguaggi più comuni

Creare report excel dal web . . . . **pag. 37**
Esportare i dati di output dalle nostre applicazioni in comodi file excel è sempre stato molto utile se non essenziale. Con le giuste librerie ora è ancora più facile. utiliziamone una, semplice e gratuita

GRAFICA

Effetti speciali per le applicazioni . . . . **pag. 37**
Impariamo come utilizzare le API di Swing-Bug per realizzare applicazioni desktop dotate di interfacce grafiche avanzate e dotate di animazioni decisamente insolite per software sviluppato in Java

Gestione dei dati con i frame . . . . **pag. 40**
Il collegamento XMLHTTPREQUEST non è il solo modo di reperire i dati da elaborare in Javascript, vedremo all'opera una tecnica che ci consente di fare a meno del lato server e di progettare applicazioni off-line

MOBILE

Grande fratello con il GPS . . . . **pag. 48**
Ecco come usare il supporto al GPS di Windows Mobile 5 ed inviare la nostra posizione ad un server Web che, tramite l'uso dei servizi messi a disposizione da Google Maps o Microsoft Virtual Earth, la traccia su una mappa

VB.NET per mobile elementi variabili . . . . **pag. 56**
Perchè utilizzare un software esterno per masterizzare i dati? Creiamo un'applicazione in grado di gestire le tracce ISO. Usiamole come se fossero una parte del nostro File System, in modo del tutto trasparente

SISTEMA

Dai una cache al tuo sotware . . . . **pag. 64**
Le applicazioni non sono mai abbastanza veloci. Le tecniche di caching sono uno dei metodi principali per aumentarne le performance. Impariamo ad utilizzare il Caching Application Block con Microsoft Visual Studio

XPS: l'alternativa Microsoft al PDF . . . . **pag. 70**
Oltre alle migliaia di funzionalità rivolte all'aspetto grafico, Windows Presentation Foundation, ci mette a disposizione una serie di servizi rivolti alla gestione dei documenti. Integriamo il nuovo XPS nel nostro software

Infrastruttura dei sistemi virtuali . . . . **pag. 104**
I nostri giorni vedono la presenza di un elevato numero di sistemi operativi dispersi in differenti versioni. Per non incorrere in problemi è importante testare il nostro software su più tipologie di installazione, ma come fare?

CORSI BASE

Posizionamento dei controlli . . **pag. 80**
Come creare applicazioni che scelgono la posizione dei componenti sulla base della risoluzione del sistema o della finestra grafica? Ecco le tecniche per non dover mai ricorrere a form non ridimensionabili

Sfuttare tutte le risorse . . . . **pag. 86**
VB.NET e gli strumenti RAD che Microsoft ha reso disponibile per la programmazione dei dispositivi mobili rendono lo sviluppo di un'applicazione un'operazione non difficile. Ma come fare per ottenere le massime prestazioni?

Gestione dei dati attraverso JDBC . . . . **pag. 92**
La rappresentazione dell'informazione è solo uno dei tanti aspetti nella progettazione di un'applicazione. Un ruolo di fondamentale importanza viene rivestito dall'interfacciamento con i database. Ecco come fare...

Diagrammi di stato . . . . **pag. 100**
Abbiamo imparato come modellare un'applicazione e a rappresentare il flusso delle operazioni. Possiamo adesso identificare in che condizioni si trova l'applicazione in un preciso istante? La risposta è si. Ecco come

SOLUZIONI

Programmazione delle attività . . . . **pag. 111**
Dovete raggiungere un obiettivo. Per farlo avete necessità di organizzare una serie di compiti ed assegnarli a diverse persone. Ci sono milioni di modi di raggiungere lo scopo. Ma qual è quello più efficace?

RUBRICHE

Gli allegati di ioProgrammo **pag. 8**
Il software in allegato alla rivista

Il libro di ioProgrammo **pag. 6**
Il contenuto del libro in allegato alla rivista

News **pag. 12**
Le più importanti novità del mondo della programmazione

Software **pag. 107**
I contenuti del CD allegato ad ioProgrammo.

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto.

http://forum.ioprogrammo.it

RIVISTA + LIBRO + CD-ROM in edicola

I contenuti del libro

PROGRAMMARE CON INTERNET

Internet nel corso del tempo ha modificato la sua fisionomia. Siamo passati da una struttura in cui i collegamenti fra i diversi contenuti presenti in rete erano realizzati con Link, ad una infrastruttura in cui tutto è interconnesso tramite API. Ad esempio è possibile localizzare la posizione di un sito tramite il suo indirizzo IP e mostrarne la locazione geografica su Google Maps. Oppure è possibile utilizzare Flickr come deposito per le immagini da visualizzare sul proprio Blog sfruttandone tutta la potenza dei tag. E ancora gli RSS consentono di integrare fra di loro contenuti diversi riaggregandoli in molteplici forme. Anche la programmazione tradizionale trae beneficio da queste innovazioni. È oggi possibile creare applicazioni che sfruttino pesantemente le migliaia di API esposte su Internet. In questo libro, ricco di esempi, vengono presentate alcune delle API più famose e viene illustrato un percorso che mette in grado il programmatore di cominciare a pensare in modo distribuito. U n Hand Book essenziale

MASHUP: OVVERO COME INTEGRARE I SERVIZI OFFERTI DA GOOGLE, MICROSOFT, TECHNORATI, FLICKR E GLI ALTRI ALL'INTERNO DELLE PROPRIE APPLICAZIONI

- **I linguaggi e la loro interazione con i Web Services**
- **Gli esempi con Google, Flickr, Technorati...**
- **Accesso diretto alle risorse Web**

Le versioni di ioProgrammo

Versione BASE

RIVISTA + CD-ROM in edicola

ZEND STUDIO 5.5.0

L'editor "ufficiale" per PHP

PHP è uno straordinario linguaggio di programmazione che ha dalla sua parte una completezza fuori dal comune, una curva di apprendimento bassissima, una velocità d'esecuzione non facile da trovare in un linguaggio interpretato. Normalmente si può scrivere codice php persino utilizzando il notepad, ma ovviamente utilizzare un editor con caratteristiche avanzate migliora di molto la velocità di coding di qualunque programmatore.Zend Studio è un editor decisamente completo prodotto dalla software house che più di ogni altra sponsorizza lo sviluppo di PHP. Le funzionalità inserite sono tantissime, dalla navigazione fra le classi, al code complexion, alla sintax highliting, al tracing e al debugging dell'applicazione.Utilizzare questo strumento migliora facilita enormemente la stesura del codice a qualunque programmatore.

Come usare l'interfaccia del CDRom

IL SOFTWARE
Una accurata recensione dei contenuti

IN EVIDENZA
Il top software del mese individuato dalla redazione

IL SOFTWARE
Il software diviso in categorie per una comoda consultazione

HOME
Torna alla pagina iniziale del CD-ROM

CONTATTACI
Vuoi inviare una email alla redazione con le tue richieste

TOP SITES
I siti più interessanti del mese selezionate per te

RICERCA SOFTWARE
Il database di tutti i software pubblicati da ioProgrammo anche gli arretrati

IL SOFTWARE
L'elenco del software contenuto nelle categorie

DIMENSIONE
La dimensione del software sul CD

SALVA
Clicca qui per installare o salvare il software sul tuo PC

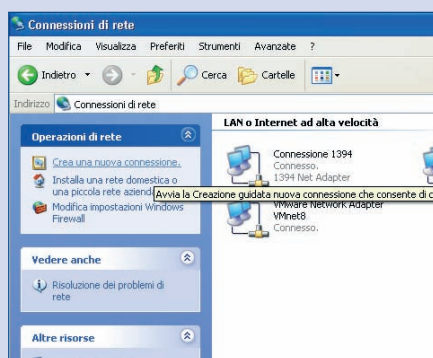
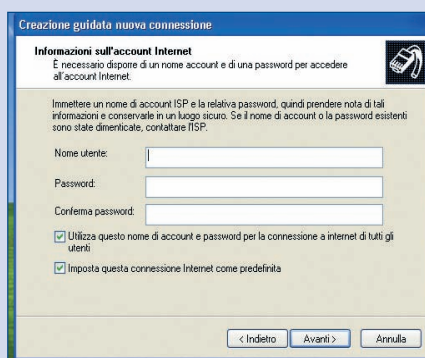
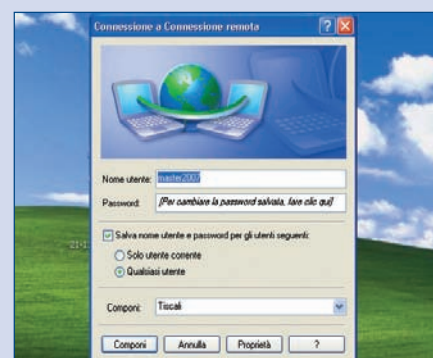
INFO
Abbonamenti informazioni e servizi utili

Online con Tiscali Easy

Numero unico per tutta l'Italia e nessuna registrazione. Il modo più semplice e veloce per entrare in Internet risparmiando

Sei spesso lontano da casa e hai necessità di collegarti a Internet ovunque ti trovi? Desideri salvaguardare la tua privacy navigando in modo anonimo? Non hai voglia di perdere tempo in lunghe registrazioni per creare un nuovo account? Niente paura, Tiscali ha pensato anche a te! Con Tiscali Easy arriva un sistema tutto nuovo di collegarsi a Internet. Non sarà più necessario creare un nuovo account e fornire i propri dati personali, potrai navigare collegandoti con un unico numero di telefono (7023456789) da tutta Italia e soprattutto utilizzando i dati di accesso forniti direttamente da Tiscali (UserID e Password presenti sulla scheda), quindi non collegabili in alcun modo a te. Insomma, con Tiscali Easy, otterrai in un colpo solo riservatezza dei dati, risparmio del tempo necessario alla creazione di un abbonamento e tariffe vantaggiose. I costi di connessione sono simili a quelli di un classico abbonamento gratuito: 1,90 cent. per i primi 10 minuti e 1,72 cent. per quelli successivi. Per risparmiare ulteriormente è possibile connettersi a Internet durante le ore serali o nei giorni festivi al costo di 1,09 cent. per i primi dieci minuti e 0,98 cent. per quelli successivi. L'unico requisito richiesto per poter eseguire la connessione è un modem correttamente installato. Poiché si tratta di una normale connessione analogica è possibile utilizzare i tool di accesso remoto integrati in Windows

IN RETE CON TISCALI

Nessuna registrazione basta creare una nuova connessione e inserire i dati di accesso forniti da Tiscali

1 NUOVA CONNESSIONE
Portiamoci nel pannello di controllo, e selezioniamo l'icona connessioni di rete. In alto a sinistra selezioniamo "nuova connessione" e prepariamoci a seguire il wizard che comparirà

2 DATI DI ACCESSO
Seguiamo il Wizard fino a quando non ci vengono chiesti i dati per l'autenticazione. E' sufficiente inserire quelli presenti nella card allegata a questo numero di ioProgrammo: master2007/tiscali

3 ACCESSO A INTERNET
Connettersi è semplicissimo, troveremo una nuova icona all'interno del pannello di controllo alla voce connessioni. Bastano due click ed il gioco è fatto sarete connessi ovunque vi troviate

News

IBM: SCIOPERO VIRTUALE

In un'epoca in cui reale e virtuale convergono in mondi paralleli è lecito attuare una protesta addirittura mettendo in sciopero i residenti del mondo di Second Life. I primi a muoversi in questa direzione potrebbero essere i dipendenti dell'IBM di Vimercate in segno di protesta verso il fallimento della trattativa per il contratto interno integrativo. E così invece di chiudere i cancelli e attuare la protesta a mezzo dello stop lavorativo, i dipendenti di IBM avrebbero scelto la via del virtuale. Lo sciopero non è però soltanto simbolico. IBM infatti risulta come una delle aziende che maggiormente ha investito nel mondo ideato da Linden. Uno sciopero dei propri dipendenti su SL significa comunque attuare una protesta significativa anche a proposito del danno economico.
In ogni caso questa vicenda sta fornendo una certa visibilità proprio agli investimenti che IBM ha fatto in questo settore. Sono proprio certi i dipendenti di IBM che non stiano, in fondo, facendo una cortesia alla propria azienda?

ADDIO PHP 4

Dopo anni e anni di gloria, il vecchio PHP4 è stato messo nel cassetto dal suo team di sviluppo. Ufficialmente il supporto verso PHP 4 continuerà fino alla fine di quest'anno ma tuttavia non ci saranno più nuovi rilasci di questa serie del linguaggio di scripting. Le motivazioni sono ovvie. PHP5 è ormai sul mercato da quasi 3 anni e PHP6 è in via di sviluppo. Il team di PHP ritiene che la versione 5 del linguaggio sia stabile e matura per cui non ci sono più ostacoli alla migrazione. Per tutti coloro che utilizzano ancora la vecchia gloriosa versione 4 è stato reso disponibile un link con una serie di istruzioni per una facile migrazione. Chi volesse consultarlo può trovarlo all'indirizzo: http://www.php.net/manual/en/migration5.php. Per la migrazione delle migliaia di applicazioni che supportano ancora PHP4 è invece utile consultare i link dei singoli sviluppatori. Raramente ne troverete di non migrabili.

UNO SGUARDO ALLA ETECH DI SAN DIEGO

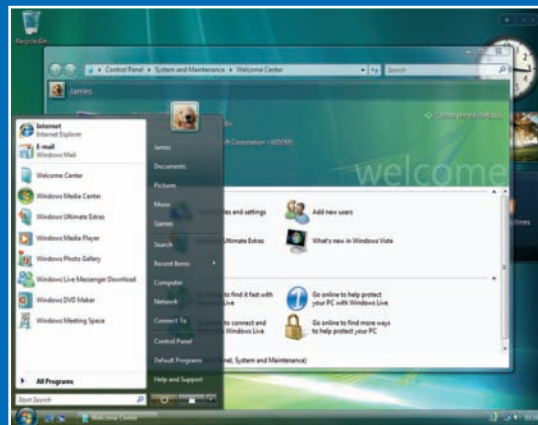
Enrica Garzilli

Conoscere il futuro è impossibile ma gettare uno sguardo sulla tecnologia futura, presentata da ricercatori d'avanguardia è che siano scienziati, sviluppatori di prodotti, capi d'azienda, profeti della tecnologia, scopritori di talenti, hacker o investitori di capitali è certamente possibile. Il 3-6 marzo 2008 a San Diego, in California, si terrà la quinta conferenza "Etech: Emerging Technologies", organizzata da O'Reilly Media (http://conferences.oreillynet.com/et2008/), che anche quest'anno si propone di discutere o di offrire la risoluzione di un problema personale, collettivo o planetario attraverso nuove soluzioni tecnologiche. è invitato a partecipare, presentando delle relazioni o tenendo dei tutorial, chiunque voglia dire in pubblico cosa sta risolvendo nella propria vita la tecnologia o cosa vuole che risolva; gli hacker che stanno lavorando a un progetto "che salverè il mondo o il tuo business, o semplicemente il tuo tempo"; chiunque voglia dimostrare in che modo la tecnologia porti alla felicità o ci scaraventi nelle situazioni che più temiamo. Gli sponsor della conferenza, oltre al padrino di Web 2.0 Tim O'Reilly e la sua quasi ventennale azienda, punto di riferimento e amplificatore dei nuovi trend tecnologici che vengono dagli "alpha geek" di tutto il mondo, sono Amazon, Adobe, Microsoft, Mozilla, Sun, Yahoo! e altri, fra cui Walt Disney. Il direttore del programma Brady Forrest ha ampliato gli argomenti proposti per il 2008 rispetto alle edizioni passate. Ha dichiarato: "Stiamo espandendo lo scopo di Etech, guardando oltre al mondo del Web a tutto quello che è manifattura, biotecnologia, sistemi su larga scala, giochi virtuali, visualizzazioni, robotica, politica, il miglioramento dell'uomo in generale e la tecnologia pulita". Questo allargherà la base dei partecipanti dai circa 1000 usuali ai previsti 1200. Gli anni passati hanno visto presentare o perfezionare progetti quali quello del Dipartimento della difesa americano, che ha creato un sistema di simulazione della terra chiamato Sentient World Simulation al fine di prevedere vari scenari possibili sul nostro pianeta come, per esempio, per quanto tempo un essere umano puè resistere senza viveri o come una persona qualunque può rispondere a un bombardamento massiccio di propaganda televisiva. SWS non rappresenta solo un mondo virtuale completo, con tutte le situazioni umane possibili, ma è calibrato secondo dati tratti dal mondo reale quali le notizie politiche piè importanti, i risultati dei censimenti, gli indicatori economici e le variazioni climatiche, insieme alle informazioni segrete di agenzie come quelle dei servizi segreti militari. Dai progetti grandiosi e istituzionale a quelli del singolo. L'anno scorso un hacker ha dato le istruzioni per creare una macchina CNC (Computer Numerical Control) a tre assi "in modo semplice ed economico", cioè per fare da soli un controllore di computer che traduce le istruzioni numeriche di un codice in movimenti per guidare uno strumento che fabbrica componenti metallici. Quest'anno, gli argomenti suggeriti per l'edizione 2008 ai potenziali relatori rappresentano delle vere e proprie sfide agli specialisti di tecnologie a qualsiasi livello. Uno dei temi è come portare avanti la causa del risparmio, della conservazione e della rigenerazione dell'energia del pianeta; un altro è il miglioramento dell'umanitè e il "body hacking", cioè cosa stanno facendo hacker, forze armate e industria privata per estendere le abilità fisiche e mentali con tecnologie miniaturizzate. Un tema decisamente provocatorio si intitola "Steal from the seedy" cioè, letteralmente, "Rubare dal degradato": cosa possiamo imparare dalla tecnologia piò avanzata del porno, delle forze armate, dei botnet, degli spammer e persino di Al Qaeda? In altre parole, come possiamo usare per scopi "puliti" quello che è nato essenzialmente come minaccia alla legalità o come anti-terrorismo informatico? Tutti i partecipanti e le idee che aiutano a prepararsi per creare un domani migliore attraverso la tecnologia sono benvenuti a Etech 2008.

SERVICE PACK 1 IN VISTA

È così siamo quasi pronti per il primo service pack ufficiale di Windows Vista. Microsoft ha schedulato il rilascio del SP1 nei primi 4 mesi del 2008. Ma già i rumors sul suo contenuto si susseguono e c'è già qualcuno che giura che sia disponibile in modo illegale sui circuiti di P2P.

Ma che cosa conterrà il nuovo service pack e quali sono le novità che gli utilizzatori e i sistemisti dovranno aspettarsi? La risposta più generica possibile è: miglioramenti dal punto di vista qualitativo, un maggiore supporto Hardware e agli standard, l'inserimento di una nuova tecnologia denominata ESO – Extensible Firmware Interface, una nuova funzione per la gestione delle tabelle d'allocazione del file system: Extended File Allocation Table. Per gli amministratori di sistema è previsto un miglioramento dei tool di deframmentazione e di quelli relativi alla diagnostica della rete.

Un significativo miglioramento in termini di sicurezza potrebbe provenire da Bitlocker, un tool per cryptare le partizioni con caratteristiche piuttosto avanzate. Il rilascio di un nuovo Service Pack a così breve tempo dall'uscita di Windows Vista ha provocato non poche reazioni nella community degli utenti. In realtà Vista si sta diffondendo a ritmi non elevatissimi anche se in modo costante. Il rilascio del nuovo SP non migliora la fiducia che gli utenti hanno nel nuovo OS. Tuttavia Microsoft non sembra essere preoccupata dalle critiche che riguardano il proprio servizio di Update. Nelle dichiarazione del management si legge entusiasmo riguardo alle tecnologie che vengono adottate per la diffusione degli aggiornamenti, e anzi il servizio viene esaltato come sinonimo di professionalità e supporto. D'altra parte, i sistemi di Microsoft, non sono i soli a subire aggiornamenti continui e l'utente dovrebbe essere ormai abituato e anzi apprezzare questa volontà delle software house di mantenere costantemente aggiornati i propri prodotti con tecniche non coinvolgono un dispendio di energie da parte dell'utente.

LA LEGISLAZIONE ITALIANA SI APRE ALL'OPEN SOURCE

In Italia si è aperto un vivace dibattito politico da quando agli inizi di giugno Bruce Perens e Richard Stallman, insieme al prof. Arturo di Corinto, hanno incontrato Fausto Bertinotti in una audizione ufficiale della Commissione cultura della Camera dei deputati (http://www.news.rai.it/dl/portal/articolonews/Page-f7277efe-5f41-42f9-983b-2c174df4c52c.html). Infatti l'Italia sta preparando un progetto di legge che prevede il rilascio di programmi informatici liberi nelle pubbliche amministrazioni e nell'istruzione. Perens, oltre ad essere uno dei principali portavoce del movimento Open Source, è consulente strategico e tecnico di Linux e Open Source Software per numerose aziende come IBM, NTT, Philips, NCR Corporation, Novell, Borland e altre piccole aziende. Stallman è un programmatore che divenne il portabandiera del free software fondando nel 1985 la Free Software Foundation (FSF), una organizzazione senza fini di lucro per lo sviluppo e la distribuzione di software libero ed è uno dei padri del concetto del copyleft. Con questo termine si indica un modello alternativo di gestione dei diritti d'autore basato su un sistema di licenze, quali la GPL per il free software o la Against DRM License per i lavori artistici (pubblicata dalla Free Creations), attraverso le quali il detentore originario dei diritti sull'opera, che di solito è l'autore, indica ai fruitori dell'opera che essa può essere utilizzata, diffusa e spesso anche modificata liberamente, pur nel rispetto di alcune condizioni essenziali.

Anche in rete e fra gli sviluppatori circola un grande dibattito sull'open source da quando il 29 giugno 2007 è stato rilasciata la versione 3 della GNU General Public License (GPL) (http://www.gnu.org/licenses/gpl.html), che regola la maggior parte dei prodotti open source. Lo scossone economico che la popolarissima filosofia della GPLv3 ha dato alle grandi multinazionali del software da una parte ne fa un nemico diretto, di cui devono tenere per forza conto e, dall'altra, ha concretizzato nel mondo dell'informatica le parole di un detto indiano: la conoscenza è una di quelle cose che, condivisa, cresce.

Enrica Garzilli
http://orientalia4all.net

SVILUPPA E VENDI SU SECOND LIFE

IL METAMONDO PROGETTATO DA LINDEN LAB HA ASSUNTO DIMENSIONI ORMAI COLOSSALI. MILIONI DI PERSONE OGNI GIORNO COMPRANO E VENDONO OGGETTI IMMATERIALI. MA CHI LI PROGRAMMA? LA RISPOSTA È UNA SOLA: VOI!

Ammettiamolo, ognuno di noi progettisti è rimasto affascinato dal mondo virtuale rappresentato nella trilogia di Matrix. I più attempati ricorderanno, inoltre, dei film “Tron” in cui un programmatore, un tal Flynn, veniva “inglobato” all’interno di un mondo virtuale e combatteva al fianco di due “personaggi” (Ram e Tron, appunto) per la propria sopravvivenza.
Ebbene ancora una volta possiamo affermare che la fantascienza può diventare realtà, infatti nell’articolo che stiamo per leggere, andremo ad esplorare un nuovo mondo rappresentato tramite realtà virtuale in tre dimensioni. Il suo nome è “Second Life” ed è raggiungibile all’indirizzo http://www.secondlife.com. Il fenomeno di Second Life è decisamente notevole e già si possono trovare delle guide da leggere, per comprendere a fondo il “nuovo mondo”.
Dato che siamo in un contesto di programmazione è bene precisare però che il contenuto dell’articolo sarà prettamente tecnico, infatti, per animare tutto ciò che è presente in Second Life, è necessario produrre script nel linguaggio di programmazione Linden Scripting Language.
In questo primo articolo affronteremo le basi del linguaggio ed analizzeremo alcuni esempi “reali”. Ma cominciamo dall’inizio.

COS’È SECOND LIFE?

Prendendo in prestito la definizione che troviamo in una apposita sezione del sito del progetto (http://www.secondlife.com/whatis), possiamo affermare che Second Life è: "Un mondo virtuale 3D interamente costruito e posseduto dai suoi residenti".
Detto in altri termini, Second Life (a cui, d'ora in poi, ci riferiremo utilizzando anche il termine "mondo") è come un immenso mondo costruito con dei "mattoncini" (ci ricordiamo dei vecchi Lego?) in cui vivono i nostri alter-ego (gli "avatar" o "residenti" che dir si voglia). Tutto ciò che vedremo nel "mondo" è costruito a partire da componenti elementari (i "prim") e reso dinamico (vogliamo dire vivo?) utilizzando il linguaggio di programmazione che vogliamo cominciare ad esplorare: il Linden Scripting Language.
Se vogliamo crearci una nuova vita virtuale, dobbiamo scaricare ed installare sulla nostra macchina un apposito programma client.
Una volta installato il client e dopo esserci iscritti a Second Life, possiamo iniziare la nostra avventura.
Come detto, in questo ambito non ci soffermeremo sui dettagli relativi alla "fruizione" del "mondo", quindi non descriveremo né i passi necessari per effettuare l'iscrizione, né tutti quegli aspetti che esulano dalla programmazione del "mondo" stesso. Tutti i dettagli di cui sopra sono dettagliatamente illustrati (in inglese) nel sito di Second Life.
Detto ciò, per entrare “in-world”, lanciamo il client quindi forniamo le nostre “seconde” generalità (nome e cognome), la nostra password, quindi premiamo sul tasto "Connect..." (si veda la Figura 1).

Figura 1: Siamo pronti per entrare in Second Life.

Benvenuti in Second Life! Ciò che vediamo è il nostro avatar, che possiamo personalizzare nell’aspetto e possiamo muovere utilizzando le frecce direzionali della tastiera. Ma non distraiamoci troppo.

LA CASSETTA DEGLI ATTREZZI

Visto che siamo qui per programmare, ci servono gli strumenti di sviluppo. Il client di Second Life contiene al suo interno tutto ciò che ci serve. Se facciamo clic sul bottone "Build" (in basso), il mouse si trasforma in "bacchetta magica"; se facciamo un clic sul terreno di fronte a noi, ci appare un cubo di legno ed il relativo dialogo di progettazione; se facciamo clic sulla linguetta "Content", ci posizioniamo nella zona in cui porre i nostri script. Facciamo clic sul bottone "New Script ...": il sistema crea per noi il primo script, che possiamo analizzare se facciamo clic sulla relativa icona, così come mostrato in Figura 2.

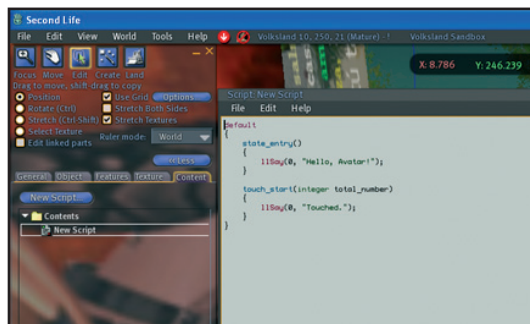



Figura 2: Il nostro primo script.

Prima di addentrarci nell'analisi del nostro primo script, dobbiamo dire due parole sul Linden Scripting Language (che indicheremo sinteticamente con la sigla LSL, d'ora in poi), il linguaggio di programmazione del mondo virtuale.

LINDEN SCRIPTING LANGUAGE

La definizione di wikipedia per il LSL è: *Linden Scripting Language is a state-event driven scripting language, in the sense of a finite state machine*. La cui traduzione ci fa capire esattamente con cosa avremo a che fare. Il LSL è un linguaggio per la programmazione di una macchina a stati finiti guidata dagli eventi. Di ogni oggetto, infatti, noi dovremo programmare la sequenza degli stati che l'oggetto potrà assumere, e gli eventi che verranno scaturiti dall/nell'oggetto. La sintassi che regola il LSL è "C-like" quindi chi di noi sviluppa già in C# o in Java non avrà nessun problema a scrivere script. Accanto ai costrutti di base (cicli, variabili, funzioni definite dall'utente, ecc.) il LSL offre una libreria di oltre trecento funzioni che ci consentono di manipolare (quasi) in ogni modo gli oggetti del "mondo". Chiaramente non avremo modo di illustrare tutte le funzioni disponibili; l'approccio dell'articolo è, invece, del tipo "hands-on-lab" ovvero analizzeremo esempi pratici per illustrare le possibilità del linguaggio in un contesto "reale".
Per capire a fondo il LSL, quindi, non ci resta che iniziare a programmare.

HELLO AVATAR!

Come per tutti i linguaggi di programmazione, anche per il LSL, il primo esempio di codice che si analizza è il classico "hello world!" che, nel contesto, assume la forma di "Hello Avatar!" (il codice è reperibile nel file Hello Avatar.lsl). Tra l'altro, a differenza degli altri linguaggi, l'ambiente di sviluppo ci aiuta, e ci compila già lo script (come mostrato nella Figura 2).

```
default
{
 state_entry()
 {
  llSay(0, "Hello, Avatar!");
 }
 touch_start(integer total_number)
 {
  llSay(0, "Touched.");
 }
}
```

Se analizziamo il codice ci accorgiamo che è definito un solo stato, *default*, che è lo stato predefinito e che deve essere SEMPRE presente in uno script LSL.
Sono definiti due gestori degli eventi: *state_entry* e *touch_start*. Il primo evento viene scatenato quando si entra nello stato *default*, ovvero quando lo script viene complato; in questo caso, dato che non vengono definiti ulteriori stati, il codice del metodo *state_entry* viene eseguito solo la prima volta. Normalmente utilizziamo il metodo *state_entry* per inizializzare

NOTA

DOWNLOAD & JOIN

Il client di Second Life è scaricabile gratuitamente e per diversi sistemi operativi, a partire dall'indirizzo http://http://secure-web11.secondlife.com/community/downloads.php mentre possiamo iscrivercial "mondo" a partire dall'indirizzo https://secureweb11.secondlife.com/join/.

EDITOR ALTERNATIVI

Sebbene il client di SL consenta da solo di sviluppare codice, è interessante dare uno sguardo alla pagina http://lslwiki.net/lslwiki/wakka.php?wakka=AlternativeEditors dove sono elencati tutta una serie di editor installabili sulla nostra macchina ed utili per continuare a sviluppare anche in assenza di connessione al "mondo". Tra questi è da segnalare LSL-Editor che, tra l'altro, fornisce un "intellisense" tipo Visual Studio, colorazione del codice, e un debugger(http://www.lsleditor.org/). Sebbene il programma non necessiti di installazione (basta scaricarlo, unzipparlo e lanciarlo), gira solo in ambiente Windows con .Net Framework versione 2.0, installato.

l'oggetto quindi per valorizzare variabili globali piuttosto che per modificare l'aspetto dell'oggetto stesso.
Restando nello stato *default*, il metodo *touch_start* viene eseguito ogniqualvolta viene toccato l'oggetto; per "reagire" al tocco utilizziamo una delle funzioni di libreria. Tutte le funzioni di libreria cominciano per "ll" (che sta per "Linden Library"); la funzione *llSay* viene utilizzata per comunicare con l'esterno (detto in altre parole per eseguire una azione di chat). Il primo parametro è il canale su cui viene inviato il messaggio che passiamo al metodo come secondo parametro. Ci sono ben 4,294,967,294 canali disponibili nell'intervallo che va da -2,147,483,648 a 2,147,483,647; il canale 0 è l'unico canale utilizzabile per mandare messaggi tra avatar e corrisponde alla chat. Tutti gli altri canali vengono utilizzati dagli script per la comunicazione tra oggetti (come vedremo più avanti).
Ripartendo dalla Figura 2, se usciamo dalla modalità di progettazione dell'oggetto (clic sulla "x" in alto a destra del dialogo di progettazione) e facciamo clic sull'oggetto, ci accorgiamo che, in effetti, nella zona in basso a sinistra appare il messaggio che ci invia l'oggetto.

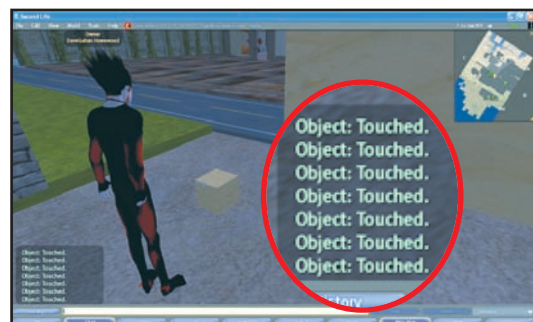



Figura 3: L'oggetto reagisce al tocco per mezzo dello script.

NOTA

WIKI SU LSL

La portata del fenomeno Second Life/Linden Scripting Language è dimostrata anche dal fiorire di siti "wiki" dove gli sviluppatori possono trovare una grande quantità di documentazione ed esempi. Tra questi sono da segnalare Second Life Wiki (http://wiki.secondlife.com/wiki/Main_Page) e LSL Wiki (http://lslwiki.net/lslwiki/wakka.php?wakka=HomePage).

UNA PORTA SCORREVOLE

Cerchiamo di approfondire la nostra conoscenza con un esempio più complesso. Anche nel mondo virtuale esistono le porte e, per aprirle e chiuderle, è necessario sviluppare del codice. Di seguito viene illustrato il codice per programmare una porta scorrevole a chiusura automatica. In questo esempio di codice (reperibile nel file *Porta Scorrevole.lsl*) cominciamo a vedere l'uso degli stati e le varie transizioni. La porta può assumere quattro diversi stati; lo stato predefinito (*default*) corrisponde alla porta chiusa. Dato che la porta non necessita di particolari configurazioni omettiamo il metodo *state_entry*. Decidiamo che la porta si deve aprire se la tocchiamo (ovvero facciamo clic sull'anta). Il metodo *touch_start* effettua una transizione di stato ponendo la porta nello stato "in apertura" (*opening*). Notiamo che sia per cambiare stato che per definire uno stato si usa la parola chiave state.

```
default
{
 // Evento scatenato dal tocco dell'avatar
 touch_start(integer total_number)
 {
  // Cambiamo stato
  state opening;
 }
}
```

Dichiariamo un nuovo stato racchiudendo tra parentesi graffe ({}), i gestori degli eventi che interesseranno il nostro oggetto nello stato che stiamo definendo.
Lo stato *opening* rappresenta l'atto dell'apertura della porta. La porta resta in questo stato solo il tempo necessario alla sua apertura. Come premessa, dobbiamo precisare che le animazioni relative al cambio di posizione degli oggetti, sono eseguite automaticamente dal simulatore di Second Life.

```
// La porta si sta aprendo
state opening
{
 state_entry()
 {
  // Ricaviamo la posizione corrente della porta
  vector pos = llGetPos();
  // Togliamo dalla componente trasversale la
                                   lunghezza della porta
  pos.y = pos.y - 2.000;
  // Imponiamo la nuova posizione
  llSetPos(pos);
  // Cambiamo stato
  state open;
 }
}
```

LE SANDBOX

Per i novizi, o comunque per coloro che non posseggono un proprio terreno in cui fare le prove, esistono delle zone franche, dette sandbox, in cui possiamo creare oggetti e provare i nostri script. Una di queste sandbox è raggiungibile al seguente indirizzo: Volksland 9, 248, 21 (gli oggetti che creiamo vengono posti nella cartella "Lost And Found" del nostro Inventory, dopo alcuni minuti di utilizzo al fine di mantenere la sandbox "pulita"; per continuare le nostre prove basta trascinare nuovamente l'oggetto sul terreno).

Per lo stato *opening* dobbiamo codificare solo il metodo *state_entry* in cui, innanzitutto, andiamo a ricavare la posizione corrente della porta attraverso la funzione di libreria *llGetPos*; la posizione è rappresentata da un vettore che esprime le coordinate cartesiane del centro dell'oggetto.
Quindi andiamo a togliere dalla componente trasversale della porta la sua lunghezza; infine settiamo la nuova posizione della porta con un altra funzione di libreria, *llSetPos*. La porta verrà spostata nella nuova posizione con un effetto "movimento" creato dal simulatore. Dopo aver imposto la nuova posizione, cambiamo nuovamente stato, ponendo la porta nello stato "aperto" (*open*).
Da notare che la componente da aggiungere dipende dall'orientamento iniziale della porta; ciò vuol dire che se orientiamo la porta lungo l'asse delle ascisse (x), la componente su cui operare sarà, appunto, la componente "x" (ed il codice verrà modificato in: pos.x = pos.x - 2.000;).

```
// La porta è aperta
state open
{
 state_entry()
 {
  // Creiamo un timer di 2 secondi
  llSetTimerEvent(2.0);
 }
// Evento scatenato dal timer
 timer()
 {
  // Cancelliamo il timer
  llSetTimerEvent(0);
  // Cambiamo stato
  state closing;
 }
}
```

Lo stato *open* ci consente di analizzare un'altra funzione di libreria: *llSetTimerEvent* che ci consente di implementare la chiusura automatica della porta; la funzione viene utilizzata per scatenare un evento di tipo *timer* ogni qualvolta trascorre l'intervallo di tempo specificato come parametro della funzione. In questo caso, dopo due secondi viene invocato il metodo timer che, in primo luogo annulla il *timer*, passando alla funzione *llSetTimerEvent* il valore zero, quindi esegue una nuova transizione di stato che pone la porta "in chiusura" (*closing*).

```
// La porta si sta chiudendo
state closing
{
```

```
state_entry()
{
 // Ricaviamo la posizione corrente della porta
 vector pos = llGetPos();
 // Aggiungiamo alla componente trasversale la
                                lunghezza della porta
 pos.y = pos.y + 2.000;
 // Imponiamo la nuova posizione
 llSetPos(pos);
 // Cambiamo stato
 state default;
}
}
```

Nello stato *closing* non dobbiamo far altro che compiere il processo inverso rispetto al metodo *opening*. Impostiamo quindi la posizione della porta al valore iniziale e ci portiamo nuovamente nello stato di *default*.

UN SEMAFORO

Per procedere nell'analisi del Linden Scripting Language, affrontiamo un esempio un po' più complesso. Finora abbiamo analizzato script associati ad un singolo elemento (*prim*); chiaramente gli oggetti più complessi necessitano una composizione di più *prim* e, di conseguenza, la programmazione dell'oggetto composito necessita di ulteriori accorgimenti. Il primo oggetto composito che analizziamo è un semaforo che è composto da un supporto più le tre lampade dei colori. La composizione grafica del semaforo è semplice; basta creare tre cilindri (o semisfere se preferiamo) e porli sopra una base rappresentata da un parallelepipedo. Per creare un unico oggetto è sufficiente selezionare ogni *prim*, quindi selezionare la voce "Link" dal menù "Tools".
È importante definire opportunamente l'ordine con cui selezioniamo i *prim* dato che l'ULTIMO selezionato prima dell'operazione di link sarà il *root prim* che regolerà il comportamento dell'intero oggetto (Per avere più informazioni riguardo agli oggetti compositi possiamo far riferimento al wiki del LSL all'indirizzo http://lslwiki.net/lslwiki/wakka.php?wakka=link).
Vogliamo programmare il semaforo affinché riproduca il classico ciclo semaforico dei segnali verde-giallo-rosso. In questo caso, vogliamo che i singoli *prim* interagiscano tra loro per sincronizzarsi a vicenda. In questo modo, il "rosso" comunica al "verde" quando accendersi; stessa cosa fa il "verde" nei confronti del "giallo" che, infine, a sua volta comunica con il "rosso".
Esistono diverse modalità di comunicazione tra prim; in alcune situazioni si utilizza la funzione

NOTA

SCRIPT LIBRARY

Una fonte molto ricca di esempi e relative spiegazioni di script in LSL è reperibile nel wiki all'indirizzo: http://lslwiki.net/lslwiki/wakka.php?wakka=ScriptLibrary

di libreria *llSay* (o *llShout* se vogliamo comunicare a distanze superiori) che abbiamo già incontrato. Chiaramente dovremo specificare un canale diverso dallo 0 usato per la chat. Normalmente questa modalità viene utilizzata per mettere in comunicazione *prim* "fisicamente" distanti come nel caso di un ascensore (che analizzeremo in un prossimo esempio).
Nel caso in cui i *prim* sono "linkati" si preferisce utilizzare la funzione di libreria *llMessageLinked* che invia messaggi solo ai *prim* che compongono l'oggetto; ciò ci mette al riparo da eventuali "interferenze" che potrebbero sorgere a causa di altri oggetti che, accidentalmente, comunicano sul nostro stesso canale.
Stabiliamo che l'accensione del semaforo viene scatenata dalla base che, in fase di avvio dell'oggetto, invia il primo messaggio ai *prim* (codice nel file *Semaforo Base.lsl*).

```
default
{
 state_entry()
 {
  // Inviamo il messaggio di "accensione rosso"
                                  all'insieme di prim
  llMessageLinked(LINK_SET, 0, "rosso", NULL_KEY);
 }
}
```

La funzione di libreria *llMessageLinked* ha diversi parametri; in questo caso il terzo parametro rappresenta il messaggio che viene inviato ovvero la lampada da accendere. Il primo parametro rappresenta il destinatario del messaggio; la parola chiave *LINK_SET* indica che il messaggio verrà recapitato a tutti i *prim* dell'insieme.
Il codice presente nelle tre "lampade" è simile; nello stato predefinito ci mettiamo all'ascolto dei messaggi "linkati" per mezzo del metodo *link_message*. Ogniqualvolta arriva un messaggio verifichiamo se è diretto al *prim* che stiamo codificando; in caso positivo attiviamo una transizione di stato, ovvero "accendiamo" la lampada. In caso negativo "spegniamo" la lampada. L'esempio di codice riguarda la lampada usata per il "rosso".

Figura 4: Il semaforo in Second Life.

```
default
{
 // Ascoltiamo i messaggi inviati all'insieme dei prim
 link_message(integer sender_num, integer num,
                                   string str, key id)
 {
  // Se il messaggio è indirizzato all'oggetto
  if(str == "rosso")
  {
   // Andiamo allo stato di accensione della lampada
   state accesa;
  }
  else
  {
   // Spegnamo la lampada
   llSetAlpha(0.0, ALL_SIDES);
  }
 }
}
```

Per "accendere" e "spegnere" la lampada usiamo la funzione di libreria *llSetAlpha* che usiamo per definire la trasparenza/opacità dell'oggetto. Quando dobbiamo spegnere la lampada imponiamo il valore del primo parametro a 0. Il secondo parametro della funzione ci serve per definire su quali facce dell'oggetto dobbiamo imporre il valore di opacità; la parola chiave *ALL_SIDES* impone il valore in ogni faccia.

```
// Stato di accensione della lampada
state accesa
{
 state_entry()
 {
  // Accendiamo la lampada
  llSetAlpha(1.0, ALL_SIDES);
  //Imponiamo la durata dell'accensione della
                                          lampada
  llSetTimerEvent(3);
 }
 timer()
 {
  // Cancelliamo il timer
  llSetTimerEvent(0);
  // Informiamo l'insieme di prim che bisogna
                              accendere il "verde"
  llMessageLinked(LINK_SET, 0, "verde", NULL_KEY);
  // Spegniamo la lampada
  llSetAlpha(0.0, ALL_SIDES);
  // Ci riposizioniamo nello stato predefinito
```

```
  state default;
 }
}
```

Nello stato *accesa*, come prima operazione accendiamo la lampada passando al metodo *llSetAlpha* il valore 1 come primo parametro. Contestualmente attiviamo un timer (*llSetTimerEvent*) che rappresenta la durata dell'accensione della lampada. Trascorso il tempo prestabilito, tramite il metodo timer dobbiamo effettuare le seguenti operazioni: annullare il timer, inviare un messaggio ai *prim* relativo alla successiva lampada da accendere, spegnere la lampada attuale, riposizionarsi nello stato di *default*.
Il codice per le diverse lampade è reperibile rispettivamente nei file *Lampada Rosso.lsl*, *Lampada Verde.lsl* e *Lampada Giallo.lsl*.
Nella Figura 4 è mostrato il nostro semaforo

UN ASCENSORE

Non tutti gli oggetti possono essere rappresentati in modo "compatto" linkando più *prim*. Nel caso di sistemi più complessi, le varie componenti possono trovarsi distribuite nello spazio come nel caso di un ascensore che è formato dalla cabina e dai pulsanti ai piani da utilizzare per chiamare l'ascensore.
Il codice associato ad ogni pulsante è molto semplice (fare riferimento ai file *Pulsante0.lsl*, *Pulsante1.lsl*, ecc.):

```
default
{
 state_entry()
 {
  // Creiamo un testo flottante al di sopra dell'oggetto
  llSetText("Chiamata",<0,0,0>,1.0);
 }
 touch_start(integer total_number)
 {
  // "Gridiamo" sul canale 999 il messaggio "1"
  llShout(999, "1");
 }
}
```

Possiamo pensare di dare un aiuto agli *avatar*, mostrando un testo flottante in corrispondenza del tasto tramite la funzione di libreria *llSetText*. Quindi dobbiamo gestire la pressione del pulsante di chiamata codificando il metodo *touch_start*; in questo caso usiamo la funzione di libreria *llShout* in modo che anche in un edificio "alto" le chiamate possano essere "ascoltate" dalla cabina. Da notare che abbiamo scelto il canale 999 per trasmettere i nostri messaggi rappresentati semplicemente dal numero del piano a cui deve portarsi la cabina.
Se passiamo ad analizzare il codice relativo alla cabina (file *Cabina.lsl*), dobbiamo gestire sia la chiamata ad un piano, che il comando inviato dall'interno della cabina, dall'avatar che sale o scende. Presupponiamo che la cabina, indipendentemente dalla sua forma, possegga una seduta che consentirà all'*avatar* che si siede, di scegliere il piano a cui portarsi. Lo script che stiamo per analizzare dovrà essere posto all'interno della seduta che dovrà essere l'ultimo oggetto selezionato prima di linkare tutti i componenti della cabina ovvero il *root prim* dell'oggetto composito.
Per rendere le cose più semplici definiamo due variabili globali: una lista che contiene le altezze dei vari piani (in modo da poter individuare immediatamente dove muovere la cabina) ed una ulteriore lista per contenere la lista dei piani. Le ulteriori variabili globali ci servono per definire la velocità con cui muoveremo la cabina, il piano e l'altezza attuale della cabina. Come annotazione, ricordiamoci che le variabili globali (che vediamo per la prima volta) vanno dichiarate in testa allo script e sono accessibili in ogni metodo di ogni stato all'interno dello script.
Nel metodo *state_entry* dello stato predefinito (*default*), tramite la funzione di libreria *llListen*, ci poniamo in ascolto dei messaggi inviati sul canale 999 (quello dei pulsanti ai piani). Su questo canale dovremo trasmettere anche i comandi della pulsantiera della cabina (che implementeremo tra un attimo). La funzione di libreria *llSitTarget*, predispone la seduta ad accogliere "opportunamente" l'avatar, accomodandolo esattamente sulla seduta e fronte verso l'esterno. L'ultima istruzione indica all'*avatar* di sedersi.

```
list ALTEZZE = [28, 29, 30, 31];
list PIANI = ["0", "1", "2", "3"];
float SPEED = 0.1;
integer piano;
integer altezza;
default
{
 state_entry()
 {
  // Ci poniamo all'ascolto del canale 999
  llListen(999, "", NULL_KEY, "");
  // Facciamo in modo che l'avatar si segga
                                        corettamente
  llSitTarget(<0,-0.5,0.5>, llEuler2Rot(<0,0,-90>) );
  // Messaggioinformativo per l'avatar
  llSetText("Sedersi prego",<0,0,0>,1.0);
```

```
}
// Metodo che riceve i messaggi inviati dagli altri oggetti
listen(integer channel, string name, key id, string message)
{
 // Verifichiamo che il messaggio proviene dal canale 999
 if(channel == 999)
 {
  // Impostiamo il piano a cui portare la cabina
  piano = (integer)message;
  // Cambiamo stato
  state moving;
 }
}
// Metodo scatenato dall'avatar che si siede in cabina
changed(integer Change)
{
 // Mostriamo all'avatar la lista dei piani
 llDialog(llAvatarOnSitTarget(), "Scegliere il piano", PIANI, 999);
}
}
```

Nello stato predefinito della cabina, dobbiamo altresì codificare il metodo di "ascolto" dei messaggi (*listen*) ed il metodo scatenato dall'avatar che si siede in cabina (*changed*). Nel primo metodo, dopo aver verificato che il messaggio provenga dal canale 999, valorizziamo la variabile globale *piano* con il valore del messaggio che contiene, appunto, il piano a cui è stata chiamata la cabina; quindi ci posizioniamo sullo stato *moving*.
Nel metodo *changed*, che si "accorge" che un avatar si è seduto in cabina, facciamo la conoscenza di una funzione di libreria che ci consente di interagire con gli *avatar*. La funzione di libreria *llDialog* mostra un dialogo ad un *avatar*, dove sono presenti diversi pulsanti che possono essere usati per consentire una scelta. La funzione accetta quattro parametri: la *key* dell'avatar a cui mostrare il dialogo, un messaggio esplicativo, una lista di opzioni (che verranno trasformate in pulsanti) ed un canale su cui trasmettere la scelta effettuata. Affinché il dialogo sia mostrato effettivamente all'avatar che si siede in cabina, usiamo la funzione di libreria *llAvatarOnSitTarget* che restituisce la *key* dell'*avatar* che è attualmente seduto sull'oggetto.
È appena il caso di notare come il canale che utilizziamo per trasmettere la scelta dell'avatar è proprio il 999 dove vengono trasmessi anche i comandi dei pulsanti a piani. In questo modo abbiamo centralizzato la gestione del movimento della cabina che, tramite il metodo *listen* appena visto, gestirà sia i messaggi provenienti dai piani che quelli provenienti dalla cabina.

```
// La cabina si sta muovendo
state moving
{
 state_entry()
 {
  // Definiamo un timer che scatta ogni decimo di secondo
  llSetTimerEvent(0.1);
  // Ricaviamo l'altezza a cui portare la cabina
  altezza = llList2Integer(ALTEZZE, piano);
 }
// Metodo invocato ogni decimo di secondo
timer()
 {
  // Ricaviamo la posizione attuale della cabina
  vector pos = llGetPos();
  // Se non siamo arrivati al piano …
  if( pos.z!=altezza )
  {
   // Se la cabina è più in alto del piano a cui portarsi …
   if( pos.z>altezza )
   {
    // Facciamo scendere la cabina
    pos.z = pos.z - SPEED;
   }
   else
   {
    // Facciamo salire la cabina
    pos.z = pos.z + SPEED;
   }
  }
  // Se siamo in un "intorno" del piano inferiore allo step di movimento …
  if(  llFabs(pos.z - altezza) < SPEED )
  {
   // Definiamo precisamente la posizione della cabina
   pos.z = altezza;
   // Azzeriamo il timer
   llSetTimerEvent(0);
   // Applichiamo la posizione alla cabina
   llSetPos(pos);
   // Informiamo l'avatar che siamo arrivati
   llSay(0,"Ascensore al piano" );
   // Ci posizioniamo nello stato predefinito
   state default;
  }
  // Applichiamo la posizione alla cabina
  llSetPos(pos);
 }
}
```

Per implementare il movimento della cabina, nel

metodo *state_entry* dello stato *moving*, definiamo un timer che scatta ogni decimo di secondo. Nel corrispondente metodo *timer*, facciamo una semplice operazione di confronto tra l'altezza attuale della cabina (tramite la componente Z della sua posizione attuale) e l'altezza a cui dobbiamo portare la cabina. La differenza tra queste due grandezze ci consente di aggiungere o togliere uno "step" di movimento (definito tramite la variabile globale *SPEED*). Dato che questa operazione viene eseguita ripetutamente ciò che ne ricaviamo è un movimento "abbastanza" fluido della cabina. Quando siamo in corrispondenza del piano, ovvero quando la differenza tra la posizione attuale della cabina e l'altezza a cui dobbiamo portarla è minore, in valore assoluto, allo "step" di movimento, vuol dire che siamo arrivati. In questa situazione azzeriamo il timer ed informiamo l'avatar che siamo arrivati.
L'implementazione del movimento appena vista ci potrebbe far sorgere una domanda: ma perché non abbiamo imposto semplicemente la posizione della cabina con il valore dell'altezza a cui dobbiamo portarla? Avremo risolto il problema con una sola linea di codice ma l'effetto collaterale è che l'animazione che il simulatore produce a fronte di una semplice variazione di posizione è estremamente rapida. Questo vuol dire che ogni volta saremmo "sparati" al piano piuttosto che "portati". Per questa ragione abbiamo dovuto parcellizzare il movimento, scomponendolo in tanti "piccoli" movimenti. Per provare i nostri script non dobbiamo necessariamente associarli agli oggetti che "in produzione" verranno animati da ciò che produciamo. Nel caso dell'ascensore, per esempio, non dobbiamo, per fora, creare un palazzo, una serie di piani, una cabina più o meno verosimile, ecc. Per provare il nostro lavoro ci è sufficiente lavorare con i "cubi" che chiunque di noi è in grado di costruire. Ad ogni cubo possiamo associare il relativo script e quindi fare le prove del caso. Nella figura 5 è presente un esempio eclatante del ragionamento appena fatto, dove l'ascensore ed i pulsanti di chiamata sono tutti rappresentati, appunto, da cubi. Il risultato grafico è tutt'altro che esaltante ma riusciamo a testare in tutto e per tutto le funzionalità che intendevamo realizzare. Questo ci consente di essere sicuri del risultato quando andremo a porre gli script negli oggetti gentilmente creati dai nostri grafici.

UNA PORTA "A CARDINI"

Facciamo un passo indietro e torniamo ad analizzare una porta; questa volta però produciamo una classica porta che ruota attorno ai sui suoi cardini. Per sviluppare questo nuovo esempio, studieremo come far ruotare i corpi in SL. Anche in questo caso useremo un oggetto composito formato dall' anta della porta e dai suoi cardini. Possiamo rappresentare questi ultimi come un minuscolo parallelepipedo che accostiamo ad uno dei lati dell'anta. Premesso che l'oggetto composito ha come *root prim*, i cardini, porremo proprio in questo oggetto il codice contenuto nel file *Porta Cardini.lsl*. Nello stato predefinito (*default*) consideriamo la porta chiusa. Per aprire la porta basta toccare la sua anta che scatena l'evento *touch_start*. Come prima operazione memorizziamo la rotazione attuale della porta nella variabile globale *rot*, quindi eseguiamo una transizione di stato che posiziona l'oggetto nello stato *opening*.

```
// Variabile globale per memorizzare la rotazione
                                    attuale della porta
rotation rot;
// Variabile globale per memorizzare la rotazione per
aprire/chiudere la porta
rotation delta;
default
{
// Evento scatenato dal tocco dell'avatar
 touch_start(integer total_number)
 {
  // Memorizziamo la rotazione attuale della porta
  rot = llGetLocalRot();
  // Cambiamo stato
  state opening;
 }
}
```

Nello stato *opening* definiamo la rotazione che deve compiere la porta quando si apre. In questo caso, usiamo la funzione di libreria *llEuler2Rot* che trasforma una rotazione espressa tramite un vettore in cui indichiamo

***Figura 5:** Uno "strano", ma funzionale, ascensore.*

gli angoli di rotazione attorno agli assi, in una variabile di tipo *rotation* che esprime la rotazione tramite una quaterna di valori (una trattazione dettagliata relativa alla gestione delle rotazioni con il LSL è reperibile all'indirizzo http://lslwiki.net/lslwiki/wakka.php?wakka=rotation).
Per definire una rotazione longitudinale (ovvero attorno all'asse Z) di 90 gradi, passiamo alla funzione un vettore con le prime due componenti a zero e con la terza componente valorizzata con la parola chiave *PI/2*. L' effettiva rotazione si calcola moltiplicando il valore ottenuto dalla funzione, per la rotazione attuale. Tramite la funzione di libreria *llSetRot*, applichiamo la rotazione che viene eseguita, come le transizioni longitudinali, in modo fluido dal simulatore.

```
// La porta si sta aprendo
state opening
{
 state_entry()
 {
  // Definiamo una rotazione di 90 gradi attorno all'asse Z
  delta = llEuler2Rot(<0.0,0.0,PI/2>);
  // Calcoliamo la rotazione sulla base della rotazione attuale
  rot = delta * rot;
  // Applichiamo la rotazione
  llSetRot(rot);
  // Cambiamo stato
  state open;
 }
}
```

Dato che anche in questo caso vogliamo dotare la porta di chiusura automatica, nello stato *open* impostiamo un timer che rappresenta il tempo in cui la porta resta aperta. Trascorso il periodo di tempo che abbiamo impostato, ci posizioniamo nello stato *close* che compie la rotazione inversa e riposiziona la porta nel suo stato predefinito (*default*).

```
// La porta è aperta
state open
{
 state_entry()
 {
  // Creiamo un timer di 2 secondi
  llSetTimerEvent(2.0);
 }
// Evento scatenato dal timer
 timer()
 {
  // Cancelliamo il timer
  llSetTimerEvent(0);
  // Cambiamo stato
  state closing;
 }
}
// La porta si sta chiudendo
state closing
{
 state_entry()
 {
  // Definiamo una rotazione di meno 90 gradi attorno all'asse Z
  delta = llEuler2Rot(<0.0,0.0,-PI/2>);
  // Calcoliamo la rotazione sulla base della rotazione attuale
  rot = delta * rot;
  // Applichiamo la rotazione
  llSetRot(rot);
  // Ci riposizioniamo nello stato predeinito
  state default;
 }
}
```

Come ultima indicazione è bene sottolineare che, come in ogni altro linguaggio di programmazione, anche con il LSL è possibile implementare una stessa soluzione utilizzando tecniche diverse. Nel caso della porta ruotante, un ulteriore esempio è reperibile nel wiki dedicato al linguaggio (http://lslwiki.net/lslwiki/wakka.php?wakka=LibraryDoor); in particolare, l'esempio propone una porta formata da un unico prim, configurata in modo da assumere un aspetto molto "verosimile", e con tanto di effetti sonori di apertura e chiusura.

CONCLUSIONI

Un nuovo mondo, una seconda vita. Questa è la prospettiva che ci attende se decidiamo di connetterci a Second Life: "*Un mondo virtuale 3-D interamente costruito e posseduto dai suoi residenti*". "Costruire", in Second Life, non significa "solo" produrre fantastiche opere in grafica 3-D; per rendere "vivi" gli oggetti di SL, è necessario produrre degli script con il linguaggio di programmazione Linden Scripting Language. Nell'articolo che abbiamo appena letto abbiamo iniziato ad esaminare le basi del linguaggio, utilizzando un approccio di tipo "hands-on-lab" in cui abbiamo potuto esaminare, attraverso degli esempi "reali", alcune delle funzioni di libreria, che ci hanno consentito di spostare, ruotare, sincronizzare gli oggetti del nuovo "mondo".

Oscar Peli

Oscar Peli (alias DaveGahan Homewood in Second Life) è .NET Solution Architect ed amministratore dei dispositivi mobili presso il Comune di Ancona. Come consulente ha sviluppato presso l'Università degli Studi di Macerata un sistema di pubblicazione di contenuti per il supporto a progetti di ricerca (http://reti.unimc.it). Oscar è contattabile all'indirizzo: opeli@unimc.it.

024-029:072-080 31-08-2007 15:52 Pagina 24

CREA IL TUO ROBOT A COLPI DI CLICK

MICROSOFT HA APPENA PRESENTATO IL SUO "VISUAL STUDIO ROBOTICS", IL PRIMO AMBIENTE RAD E VISUALE PER LO SVILUPPO RAPIDO DI APPLICAZIONI DESTINATE ALL'AUTOMAZIONE ROBOTICA. ECCO COME INSTALLARLO E USARLO FACILMENTE

Tutto nasce da un articolo di Bill Gates sull'autorevole rivista americana Science intitolato "A robot in every home"…
Il Chief Architect di Microsoft racconta che, negli ultimi anni, in occasione di visite alle principali Università americane, tra i progetti che gli venivano presentati, ve ne era costantemente almeno uno dedicato all'automazione ed alla robotica…
Bill Gates ha riflettuto su tutto questo e con un audace volo di fantasia ha ritrovato delle somiglianze tra l'attuale scenario della cibernetica e quello dell'informatica, all'epoca in cui lui e Steve Allen muovevano i primi passi…
Per essere protagonista anche di questa rivoluzione prossima ventura ha promosso il rilascio una serie di software adeguati alla programmazione dei Robot;
Visual Robotics Studio è appunto l'ambiente di programmazione che Microsoft dedica alla cibernetica;
Il colosso di Redmond ha seguito una filosofia di sviluppo analoga a quella già implementata con l'XBox, assemblando in maniera originale tecnologie che Microsoft conosce bene;
il risultato è una specie di Visual Basic che al posto della tipica toolbox riunisce, come in una catena di montaggio, pezzi di robot mettendo in gioco anche i web services, uno dei cavalli di battaglia della prima release di .net, e XNA il Kit di sviluppo dedicato ai videogiochi della XBox 360°, etc
Vediamo nel dettaglio come lavora questo All Star Team

VISUAL PROGRAMMING LANGUAGE

Il primo Tool che prendiamo in esame si chiama Visual Programming Language;
attraverso una interfaccia grafica, che funziona come una lavagna, disegniamo una sorta di catena di montaggio che collega e fa funzionare i diversi componenti di una automazione;
L'ambiente grafico è raffigurato in Fig 1:

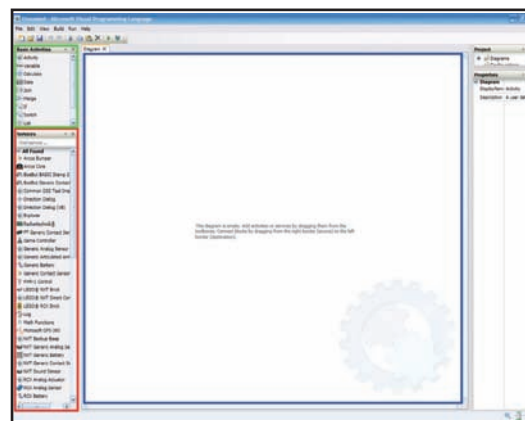



Figura 1: L'ambiente di programmazione di Visual Studio Robotics

SERVICES

Evidenziata in rosso troviamo la finestra Services; è un elenco di 'pezzi di Robot':
scorrendoli scopriamo oggetti come i mattoncini LEGO MindStorm e Fishertechnics, joystick per videogiochi, ma anche hardware semi professionale come il MobileRobot Pioneer fabbricato da MobileRobots od i componentistica della Parallax;
questi 'pezzi di Robot', o per essere tecnici 'Services', rappresentano l'hardware ai quali fanno riferimento, ne avvolgono la complessità e la astraggono, rendendola trasparente allo sviluppatore;
per iniziare a lavorare con i Services non abbiamo che da clickarvi sopra e trascinarli nella finestra centrale Diagram, evidenziata in blu;
è inevitabile notare le somiglianze con i tipici ambienti di sviluppo di Microsoft:
clickare sopra un servizio e trascinarlo nella finestra Diagrams è una procedura analoga alla creazione di pulsanti, Textbox, eccetera, nelle WebForm e WindowsForm, ovvero un paradigma di programmazione presente sin dalla primissima release di Visual Basic!
Non solo…
Le specifiche dei "Pezzi di Robot" sono descritte in file Manifest, ovvero in file di configurazione in

Conoscenze richieste
Conoscenza degli ambienti di programmazione Microsoft .net

Software
Microsoft Robotics Studio 1, .NET Framework 3, Visual Studio 2005 o Kit Visual Express C# o VB.net ad libitum

Impegno

Tempo di realizzazione

formato XML;
i file Manifest sono forniti principalmente dai fabbricanti di Robot e, con l'andare del tempo, Microsoft conta di coprire ogni tipo di Hardware disponibile; I Services, con i relativi file Manifest, forniti dai fabbricanti di hardware ci ricordano senza dubbio i vari componenti sviluppati da terze parti disponibili per Visual Studio.

BASIC ACTIVITIES

Una volta disposti i 'pezzi di robot', o per usare il termine tecnico i Services, è necessario fornirgli delle istruzioni e farli dialogare tra loro;
le istruzioni vengono allocate e manipolate attraverso una serie di oggetti denominati "Basic Activities" disponibili nella finestra evidenziata in verde. Le Basic Activities sono rappresentate graficamente attraverso 'quadratini' con segmenti verdi sui lati sinistro e destro; il nome in codice di questi segmenti verdi è 'pin activities'; nel diagramma della nostra automazione i dati che 'rendono elettrici' i vari Services entrano nel pin posto a sinistra del 'quadratino' ed escono, dopo essere stati elaborati ed instradati, dal pin posto a destra;
ogni finestra "Basic Activities" possiede Wizard di configurazione per impostare semplicemente le sue proprietà; Vediamo le varie 'Basic Activities' nel dettaglio

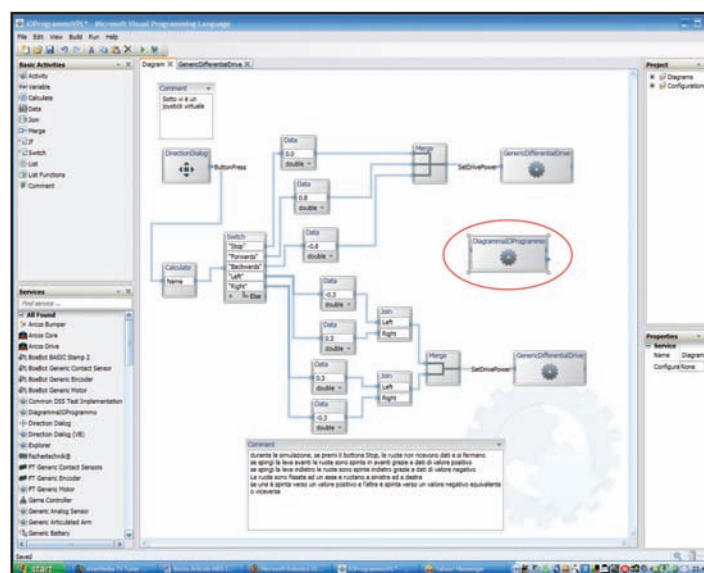
- **Variabile** Crea una variabile e la tipizza; le variabili sono tipizzate attraverso i medesimi tipi previsti in C# (cfr. Box dedicato)
- **Data** Alloca i dati della nostra automazione in variabili (ad esempio inserisce "Hello World" in una variabile di tipo String)
- **List** Crea e tipizza una serie di dati (nota che è possibile creare una lista di dati anche con 'Variable')
- **List Functions** Consente di modificare un List pre esistente
- **Calcolate** Consente di sommare, sottrarre dividere o moltiplicare variabili numeriche oppure di concatenare stringhe
- **If** Questa Activity, in maniera analoga ad un tipico ciclo condizionale If/ Else, sposta il flusso dell'automazione verificando se una condizione è vera oppure è falsa
- **Switch** In maniera analoga all'Activity sposta il flusso dell'automazione ma a differenza di questo smista il flusso confrontando il messaggio in ingresso con un valore impostato dallo sviluppatore
- **Merge** 'Fonde' semplicemente tra loro due o più flussi di automazione
- **Join** Unisce due o più flussi di automazione dando ad ognuno di questi uno specifico nome
- **Activity** È possibile racchiudere l'intero flusso di automazione descritto in un diagramma i una 'Activity'; l'Activity viene compilata selezionando la voce Run dai menù a tendina (cfr. Fig. 2 e didascalia)
- **Comment** Inserisce un commento esplicativo al flusso di automazione; non influenza il programma ma lo chiarisce ai nostri occhi

Per finire, a sinistra del tool VPL troviamo le finestre

Project
che, in maniera analoga alla finestra Project di Visual Studio, ci consente di impostare il nome del Diagramma, di aggiungere oppure o cancellare un diagramma

Properties
che consente, in maniera analoga alla finestra Properties di Visual Studio, di impostare le proprietà dell'oggetto Services oppure Activity selezionato.

IL PROGETTO D'ESEMPIO

Questo è il diagramma, basato su quelli allegati all'articolo che implementa l'automazione di un robot comandato da un Joystick; sono messe in gioco tutte le Basic Activities di cui abbiamo parlato e diversi Services i cui file Manifest supportano il Simulatore; In breve l'automazione funziona in questo modo: durante la simulazione, se premi il bottone Stop, le ruote non ricevono dati e si fermano se spingi la leva avanti le ruote sono spinte in avanti grazie a dati di valore positivo spingi la leva indietro le ruote sono spinte indietro grazie a dati di valore negativo Le ruote sono fissate ad un asse e ruotano a sinistra ed a destra se una è spinta verso un valore positivo e l'altra è spinta verso un valore negativo equivalente o viceversa.
Nota il Services di tipo Activity che incapsula tutto il flusso di Automazione del nostro Diagramma ed i vari commenti esplicativi. Il disegno, o per usare il termine tecnico, il diagramma che creiamo ricorda concettualmente un diagramma UML oppure anche ad tipico diagramma di flusso ma non coincide affatto con questi formalismi

Visual Studio Programming viene fornito con una serie di diagrammi esemplificativi;
ho preso il quarto di questi esempi e l'ho modificato per renderlo più calzante ai nostri scopi "didattici".

ROBOT & REALTÀ VIRTUALE

Questo paragrafo risponde alla domanda: "Come i diagrammi di VPL riescono a diventare un'automazione?"
Una volta assemblata l'automazione, clickando sull'icona Play, anch'essa tipica degli ambienti di sviluppo Microsoft, è possibile telecomandare un robot, oppure avviare una animazione che simula l'automazione, quasi fosse un videogioco.
Sono i files Manifest dei diversi Services che impostano il simulatore grafico piuttosto che l'hardware; per intendersi, il simulatore viene avviato quando l'automazione utilizza Services le cui caratteristiche sono descritte in file Manifest che richiedono il simulatore grafico mentre automazioni assemblate con Services i cui Manifest richiedono hardware comunicano e richiedono la connessione con un Robot vero e proprio (cfr. Fig. 3 e relativa didascalia);
Nel nostro caso, poiché non disponiamo di alcun Robot, è necessario lanciare il simulatore grafico, nome in codice Microsoft Visual Simulation Environment

L'AMBIENTE DI SIMULAZIONE

Dalla Finestra Diagram del VPL è possibile selezionare quale File Manifest intendiamo utilizzare per i nostri Services: Se utilizziamo un File Manifest dedicato ad Hardware reale il Framework cercherà i pezzi restituendo un messaggio di errore se non li trova (non ho il mattoncino Lego Mindstorm e non viene intercettata alcuna connessione Bluetooth) Se utilizziamo invece un Manifest dedicato ad una Hardware simulato viene lanciato automaticamente il Visual Simulation Environment (un robot Lego Mindostorm inizia la sua esplorazione nel mondo virtuale realizzato con il game engine di AGEIA e XNA); Nota bene che se avessimo il file Manifest relativo potremmo pilotare, dal nostro computer, i robot Spirit ed Opportunity che hanno esplorato Marte

MICROSOFT VISUAL SIMULATION ENVIRONMENT

Se, testando le vostre applicazioni con il simulatore, avete l'impressione di giocare con un videogames non siete poi così lontani dalla realtà…
La tecnologia impiegata per il Visual Simulation Environment è appunto quella del Game Engine PhisX della software House Ageia.
ricordo brevemente che un Game Engine è un software "a basso livello", utilizzato nella programmazione dei videogiochi, che si occupa della grafica, delle leggi fisiche, delle animazioni e dell'intelligenza artificiale, etc fornendo un modello coerente sul quale lo sviluppatore può lavorare.
Questo Game Engine è programmato utilizzando un ambiente di sviluppo che dovrebbe essere noto ai lettori di IoProgrammo: XNA Game Studio di Microsoft, che viene utilizzato principalmente nella programmazione della console XBOX 360°…
Lo spazio di un articolo è sempre troppo breve e non mi è possibile sviscerare tutte le funzionalità del simulatore che sono comunque piuttosto intuitive;
vale la pena però spiegare brevemente come è possibile personalizzare lo scenario delle nostre simulazioni..
È possibile aggiungere, posizionare o eliminare le figure geometriche che popolano lo scenario della simulazione entrando nella modalità Edit attraverso la pressione del tasto funzione F5;

Figura 4: Nota bene che è possibile creare nuove entità per popolare lo scenario importandole da un ambiente di modellazione grafica evoluto quale 3d studio max!

confrontando la Fig. 4 scopriamo che a sinistra della tipica finestra del simulatore si va ad aggiungere un'area che

- Enumera tutti gli elementi che popolano la simulazione (evidenziata in rosso)
- Consente di posizionare un elemento selezionato secondo i tre piani cartesiani ed anche di ruotarlo (evidenziata in blu)

È possibile modificare gli elementi geometrici della simulazione seguendo questa procedura:

a. Selezionandoli nell'area evidenziata in rosso
b. Dopo, attraverso il menù a tendina Entity, con le tipiche operazioni di Copia & Incolla è possibile cancellarli, raddoppiarli oppure eliminarli.

Dal menù a tendina File è possibile salvare uno scenario (selezionando Open Scene…) e caricarne uno (selezionando Save Scene as…);
negli esempi allegati all'articolo troverete anche uno scenario dedicato all'automazione di esempio che vi consentirà di guidare il robot fatto con i Lego in un percorso a zig a zag attraverso una sequenza di coni stradali.
Nota, per concludere, che è anche possibile modificare la gravità del nostro scenario, un'opzione che si può rivelare utile qualora dovessimo progettare l'automazione per un robot esploratore su Marte o sulla Luna.

PROGRAMMAZIONE DEI SERVICES

Discutendo con l'Ing. Manfrin sul Kit di sviluppo l'attenzione è stato soprattutto focalizzata sul concetto di Orchestration:
con "Orchestration" intendiamo la perfetta sincronia, dall'inizio alla fine, di tutti i Services che compongono la nostra automazione;
per intendersi: nella nostra simulazione non succede nulla se il nostro robot, fatto con i Lego virtuali, urta un ostacolo per una disattenzione, mentre nella realtà gli errori casuali o le imprecisioni di una macchina utensile possono causare danni incalcolabili, pensate alla catena di montaggio di una fabbrica di automobili i cui robots, scoordinati, smaltano, saldano, assemblano pezzi che non sono ancora arrivati o sono sistemati male.
L'Orchestration è un problema assimilabile ai problemi classici di sincronizzazione dei Threads nei Sistemi operativi, esemplificati con celebri rompicapo quali la "Torre di Hanoi" piuttosto che la "Cena dei Filosofi;
problemi molto complessi, infernali che sono parte degli incubi degli informatici quasi da sempre…
Programmare un'applicazione con Microsoft Robotics significa dunque gestire la difficile Orchestration tra gli input e gli output che si scambiano i Services;
L'Orchestration viene gestita da una nuova libreria software che è probabilmente la punta di diamante del Pool di software che compongono questo originalissimo Framework:
Concurrency and Coordination Runtime (CCR) che tenta di risolverla rendendola invisibile ai nostri occhi, ovvero gestendola dietro le quinte.
Nota che Bill Gates nel sopraccitato articolo sulla rivista Science nomina esplicitamente il CCR, ad un pubblico non specializzato, spiegando appunto che risolve (sic…) il problema del MultiTasking di diversi processi paralleli

IL CCR VISTO DA VICINO

Prima di addentrarci nei dettagli di questo componente software è necessario capire bene che cosa fa: immaginate di dovere creare una intranet per una LAN composta da 100 computer e dover verificare se tutti sono online: probabilmente scrivereste un componente che lancia un ping verso ogni singolo computer e ne attendi la risposta, gestendola poi per i vostri scopi.
Utilizzando la libreria CCR non avreste che da dirgli che cosa fare (lanciare un ping) e come gestire le risposte; ogni singolo ping viene gestito dal CCR creando per ogni IP un thread che utilizzerà la scheda di rete in più richieste parallele.
Enunciato in questi termini la libreria pare essere proprio essere una cosa notevole.
la realtà però non è così rosea.. Il CCR, visto da vicino ha un aspetto piuttosto ostile..

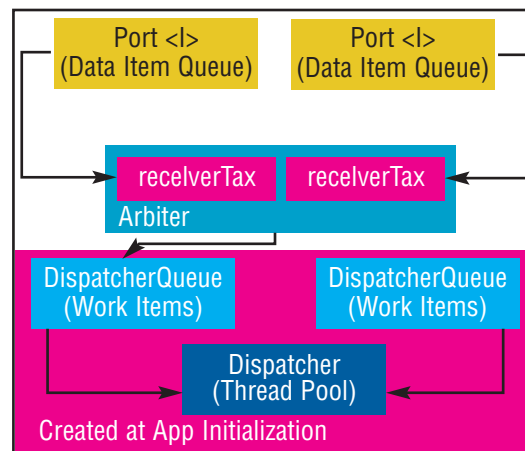



Figura 5: Lo schema della CCR

La libreria CCR (cfr. Fig. 5) è scritta in C# e sfrutta alcune caratteristiche della nuova release del Framework.net;

LA CLASSE PORT

Il CCR innanzitutto definisce una classe di tipo Port che riceve in ingresso i parametri che dovranno essere elaborati tipizzandoli;

Ad esempio questo pezzo di codice

```
Port<float> pF = new Port<float>() ;
```

significa che abbiamo istanziato una classe Port di nome pF che riceve in ingresso un parametro di tipo Float (nota le parentesi angolari);
è possibile ingressare 16 parametri di tipo diverso;
con questa sintassi, ad esempio

```
PortSet<int,string> pIS = new PortSet<int,string>();
```

passiamo due parametri di tipo integer e string.
La classe di tipo Port comprende una struttura di dati FIFO (il dato che entra per primo (First In) è il primo dato che viene elaborato e restituito (First Out)); i dati accodati nella porta vengono poi "affidati" ad un arbitro che a sua volta chiama, facendo uso di delegate e metodi anonimi, una funzione da eseguire su questi dati; per la precisione i dati sono elaborati chiamando un delegate (un costrutto tipico di .net, simile alle chiamate a funzioni del C), che viene associato ad un metodo denominato oppure, ed è questa la novità, un metodo anonimo; i metodi anonimi sono una novità di C# e, in maniera elegante consentono di passare un blocco di codice come parametro del delegate. Ad esempio:

```
// Crea un gestore per un evento Click
button1.Click += delegate(System.Object o,
                          System.EventArgs e)
        {
 System.Windows.Forms.MessageBox.Show("Click!");
        };
```

I metodi supportati dalla classe Port sono
Post che invia un messaggio ad una porta

```
Port<int> pi = new Port<int>() ;
 pi.Post (42) ;
```

Test che verifica, attraverso un valore Booleano di ritorno, se la porta è occupata

```
int iv ;
 if (pi.Test (out iv))
   Console.WriteLine ("Read " + iv) ;
 else
   Console.WriteLine ("Port empty.") ;
```

LA CLASSE ARBITER

Una volta che i messaggi sono passati attraverso la classe Port e necessario elaborarli e smistarli; questo compito è svolto dalla classe Arbiter che li elabora utilizzando un delegate, che chiama la funzione specifica, e poi li invia alla classe Dispatcherqueue che li smista. Brevemente i metodi della classe Arbiter, ordinati dai più semplici verso i più complessi

```
public static class Arbiter {
    public static ITask FromHandler(Handler
 handler);
```

è un metodo che specifica che un valore può essere accodato verso una classe DispatcherQueue (che vedremo più avanti).
Questo semplice membro della classe Arbiter non è associato ad alcun membro della classe Port

```
    public static ITask FromIteratorHandler(Handler
                                         handler);
```

è un metodo che enumera diversi valori che possono essere accodati verso una classe DispatcherQueue che vedremo più avanti.
Non è associato ad alcun membro della classe Port

```
    public static Receiver<T> Receive<T>(
        Boolean persist, Port<T> port, Handler<T>
                                         handler);
```

è un metodo che può essere chiamato per passare il risultato di una elaborazione verso una singola porta. Deve essere specificato sia il delegate che chiamerà l'evento sia la porta.

```
    public static JoinSinglePortReceiver
                           MultipleItemReceive<T>(
        Boolean persist, Port<T> port, Int32
                                       itemCount,
        VariableArgumentHandler<T> handler);
```

è un metodo che può essere chiamato per inviare diversi risultati dell'elaborazione verso una singola porta.
Il parametro inviato al delegate VariableArgumentHandler consiste in un array di oggetti Port

```
    public static JoinReceiver MultiplePortReceive<T>(
        Boolean persist, Port<T>[] ports,
        VariableArgumentHandler<T> handler);
```

è un metodo che può essere chiamato per inviare un singolo risultato dell'elaborazione verso molteplici porte. Gli item devono essere del medesimo tipo delle porte. Il parametro inviato al delegate VariableArgumentHandler consiste in un array di oggetti Port

```
    ... // ed altri membri omessi per brevità
 }
```

LA CLASSE DISPATCHERQUEUE

il termine queue significa coda ed identifica la coda delle richieste dell'elaborazione che la classe Arbiter desidera siano smistati dalla classe Dispatcher;
I threads gestiti nella classe del Dispatcher sono in attesa degli input accodati nella classe DispatcherQueue

```
public sealed class DispatcherQueue : IDisposable {
    /* Usa il pool di thread gestito dal CLR non quello
                                        del Dispatcher più
flessibile */
    public DispatcherQueue(string name);
    public DispatcherQueue(String name, Dispatcher
                                             dispatcher);
    public void Dispose();
    ... // altri membri omessi per brevità
}
```

LA CLASSE DISPATCHER

Il compito di sincronizzare i vari thread dell'elaborazione viene affidato alla classe Dispatcher.
La classe Arbiter ha accodato alla dispatcherqueue delle funzioni da eseguire. La classe Dispatcher trova nella queue che funzioni ci sono e le esegue attraverso il Multithreading, cioè in parallelo.

```
public sealed class Dispatcher : IDisposable {
    public Dispatcher();
    public Dispatcher(Int32 threadCount, String
                                        threadPoolName);
    public Dispatcher(Int32 threadCount,
                                   ThreadPriority priority,
        String threadPoolName);
    public ICollection<DispatcherQueue>
                                 DispatcherQueues { get; }
    ... // altri membri omessi per brevità
}
```

Per default ogni thread viene indirizzato ad ogni singola CPU presente nel computer (nota che in questo modo sono ottimizzati i moderni processori Multi Core) ed esternamente è possibile identificare ogni singolo thread attraverso una stringa e settarne il livello di priorità (per default, il Framework assegna ad ogni thread un nome che viene poi utilizzato nel Visual Studio® debugger's Threads). È possibile creare molteplici classi Dispatcher, ognuno con il suo pool di threads amplificando indefinitamente le possibilità di elaborazione dello stesso. Per concludere avrete certamente compreso che Il CCR è un componete ostico da programmare e probabilmente vi sarete resi conto che gestire l'orchestrazione tra i vari Services sarebbe ben più difficile che non costruire a mano il proprio Robot… Ebbene le buone notizie sono che grazie a Microsoft Robotica Studio, non avrete da scrivere una sola riga di codice; l'intera implementazione del CCR viene gestita attraverso il VPL che genera tutto il codice del quale abbiamo bisogno senza doverci mettere le mani; La documentazione del CCR per ora è veramente scarsa ma le potenzialità di questo componente sono tali che, dovrebbe entrare a fare parte integrante delle prossime release di .net

CONCLUSIONI

Microsoft, ogni volta che si trova ad affrontare una sfida in un settore tecnologico differente dal suo 'Core Business', non parte dal nulla ma riprende, rielaborandoli, diversi suoi cavalli di battaglia; la cosa è evidente in XBox è soprattutto in questo Robotic Development Kit che ci presenta sotto una luce nuova non poche vecchie conoscenze… Tra queste la vera novità e la libreria CCR che molti sviluppatori vorrebbero ripreso nella Release ufficiale del .NET Framework; Abbiamo parlato di robots e posto l'accento sulle comunità di appassionati che li smanettano ma il VRS non è un curioso Gadget quanto piuttosto lo strumento che programma le macchine utensili e le catene di montaggio; è appena uscito dalle officine Microsoft ed a non poche carenze, quantomeno nella documentazione, ma ha caratteristiche davvero interessanti che certamente ritroveremo nel nostri programmi anche, nel bene o nel male, ci occupiamo tutto altro che di costruzioni Lego.

Luigi Corrias

RINGRAZIAMENTI
Per la preziosa collaborazione si ringrazia l'ing. Federico Manfrin http://federicomanfrin.googlepages.com che si è prestato con la sua esperienza ad aggiungere note importanti a questo articolo.

COME SI AVVIANO I SERVIZI?

I Services sono creati, avviati monitorati e distrutti grazie ad un componente software il cui nome in codice è Decentralized Software Services Node;
Il nodo DSSN si occupa anche di farli comunicare attraverso un protocollo basato su SOAP: DSSP; (nota che la l'idea della comunicazione tramite SOAP è ripresa da uno dei pilastri di .NET, i Web Services che si scambiano dati appunto su protocollo SOAP e WSDL)
Per creare un Services utilizzando DDS non abbiamo che da lanciare l'Interfaccia a linea di comando del Kit di sviluppo e scrivere:

```
dssnewservice /s:NomeServizio
                        /l:Linguaggio
```

e generare, nella cartella di installazione del Kit, un vero e proprio progetto di Visual Studio 2005, completo di certificato, pronto per essere lanciato e programmato(cfr. Fig. 6)!
Per comodità io ho scelto di utilizzare VB.net e di gestire il progetto con Visual Studio 2005 ma, nota bene, questo ambiente, se non è utilizzato a scopi commerciali, è completamente gratuito e può essere programmato altrettanto bene con i vari Visual Studio Express;

TRASFORMAZIONI XSLT LATO SERVER

XSLT CONSENTE DI OTTENERE UN DOCUMENTO IN UN QUALUNQUE FORMATO A PARTIRE DA UNA BASE XML. AD ESEMPIO DAGLI STESSI DATI SI PUÒ OTTENERE UN FILE .PDF OPPURE UN FILE .DOC. IMPARIAMO COME INTEGRARE QUESTA TECNICA CON I LINGUAGGI PIÙ COMUNI

Conoscenze richieste
elementi di Java per il web, PHP5, e Classic Asp, basi di XSL

Software
un browser, IIS, Apache, un SDK recente di Java, PHP5, Classic Asp

Impegno

Tempo di realizzazione

Già nel passato, in questa stessa rivista, si è molto parlato di XML, con un particolare risalto agli aspetti più pratici e concreti che caratterizzano questo importantissimo formato.
Si è visto che XML offre grandi vantaggi per lo sviluppatore, a cominciare dalla sua particolare struttura, semplice e ordinata, che fornisce un'interfaccia “standard” per la gestione e l'organizzazione dei dati, in grado di far conciliare con successo anche tecnologìe o protocolli estremamente diversi tra di loro.
Oltre ad essere ormai diventato una sorta di “lingua franca” per l'informatica, utilizzata da tutte le principali tecnologìe per comunicare vicendevolmente, XML offre poi qualità di immediatezza e semplicità interpretativa davvero eccezionali, che lo rendono il formato di conservazione dei dati ideale anche per i “non addetti ai lavori”.
L'utilizzo di XML, in particolare, ci consente di “disaccoppiare” con facilità la fase di sviluppo applicativo (ossia quella legata all'estrazione e all'elaborazione intermedia dei dati), di competenza esclusiva del programmatore, da quella meramente rappresentativa, di cui invece si occupa il *web-designer*: ciò avviene utilizzando **XSL**, un linguaggio *XML-based* che, tramite un programma apposito detto **processore XSLT**, è in grado di trasformare un file XML in un file di tutt'altro formato (SVG, XHTML, ecc...).
Nel passato abbiamo, in particolare, analizzato un caso di “**trasformazione XSLT lato-client**”, dove, cioè, la fase di elaborazione grafica e stilistica dei dati avveniva direttamente sul browser. Tale soluzione era caratterizzata dal fatto che il server si limitava a consegnare al client un semplice file XML con i dati richiesti, e non già, quindi, il codice XHTML risultante dall'elaborazione degli stessi.
Se i vantaggi di questo approccio sono legati ad una diminuzione del traffico di rete (visto che l'”oggetto di scambio” tra client e server è un banale file XML, invece che un codice XHTML completo), e ad una riduzione del carico di lavoro lato-server, gli svantaggi sono però dovuti alla necessità di dover disporre sul client di componenti idonei ad effettuare detta trasformazione.
Chi ha già familiarità con lo sviluppo lato-client con Javascript immaginerà facilmente, a questo punto, quanto un simile approccio sìa poco realistico se riferito ad un ambiente di produzione, dove cioè i potenziali utilizzatori della *web-application* non necessariamente sono provvisti di tutti i requisiti *software* richiesti.
L'unica soluzione, pertanto, se l'obiettivo è quello di garantire la massima fruibilità del prodotto, è spostare la fase di elaborazione XSLT direttamente sul server.
Va precisato, a questo punto, che il presente articolo non si prefigge di trattare in modo completo ed esaustivo l' “argomento XML” nei principali linguaggi lato server, visto che sicuramente non basterebbero diversi volumi.
Ci limiteremo, piuttosto, ad analizzare gli strumenti che ci vengono messi a disposizione dai principali linguaggi lato-server, per l'effettuazione di una semplice trasformazione XSLT.
A questo riguardo, utilizzeremo il file XML sotto riportato:

```
<?xml version="1.0" encoding="UTF-8"?>
<progetto>
  <fase>
    <descrizione>fase 1</descrizione>
    <contenuto>
            <fase>
            <descrizione>fase 1_1</descrizione>
            </fase>
            <fase>
            <descrizione>fase 1_2</descrizione>
            </fase>
            <fase>
            <descrizione>fase 1_3</descrizione>
            <contenuto>
                        <fase>
<descrizione>fase 1_3_1</descrizione>
```

```
                    <contenuto>
                    <fase>
<descrizione>fase 1_3_1_1</descrizione>
                    </fase>
          ...
          ...
          ...
</progetto>
```

Si tratta, evidentemente, di un semplice file XML che descrive la composizione di un progetto, suddiviso in più fasi, le quali a loro volta contengono ulteriori sotto-fasi, ecc...
Tramite il file XSL seguente, siamo in grado di modellare la gerarchia delle fasi facenti parte del progetto in questione:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

  <xsl:template match="progetto">
  <html>
  <body>

    <xsl:apply-templates select="fase" />

  </body>
  </html>
  </xsl:template>

  <xsl:template match="fase">
    <ul>
      <li>
        <xsl:value-of select="descrizione" />
        <xsl:apply-templates select="contenuto/fase"
/>
      </li>
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

Trascuriamo, in particolare, tutte le problematiche relative alla creazione dei modelli DTD o degli XML Schema, utili per effettuare la validazione della definizione semantica dei file XML, limitandoci a farne qualche accenno in un box a parte.
Come possiamo vedere, il file XSL parte dal nodo principale (“progetto”), e quindi definisce un **template**, ossìa una “regola” che il processore XSLT (il programma che deve effettuare la trasformazione) deve applicare tutte le volte che incontra il nodo “fase” durante il *parsing* del file XML: nel caso in questione, verrà generato un tag HTML UL (lista non ordinata), che contiene come elementi la descrizione della fase e le eventuali altre sotto-fasi in essa presenti. In altre parole, il template è applicato in modo ricorsivo per ricostruire l'intera gerarchia delle fasi contenuta nel file XML d'origine.

EFFETTUIAMO LA TRASFORMAZIONE

In precedenza abbiamo introdotto i due file XML (il contenitore dei dati da trasformare) e XSL (il modello a partire dal quale verrà effettuata la trasformazione). A questo punto, non ci resta che realizzare il “motore” lato-server che deve eseguire materialmente l'elaborazione XSLT.
Cominciamo con Asp (Active Server Pages), che utilizzeremo nel modo più classico, seguendo la logica di programmazione procedurale che maggiormente la caratterizza:

```
<%
 'Dimensioniamo due variabili, che dovranno
                          contenere i riferimenti
' ai nostri due file

 Dim objXML
 Dim objXSL

 set objXML =
        Server.CreateObject("Microsoft.XMLDOM")
 objXML.async = false

 set objXSL =
        Server.CreateObject("Microsoft.XMLDOM")
 objXSL.async = false

 'carichiamo il file XML
 objXML.load (Server.MapPath("progetto.xml"))
```

COME SI INSTALLANO GLI ESEMPI ?

Ecco le indicazioni per installare i file di esempio:
Asp: basta copiare la cartella relativa nel webserver (tipicamente IIS), e inserire i due file XSL e XML nella stessa;
PHP: occorre copiare la cartella con i sorgenti nel webserver inserendovi anche i due file da processare;
Jsp: se si usa Tomcat come Application Server, è sufficiente copiare il file .WAR nella cartella Webapps, ove risiedono in generale le web-application.

N.B. Per far funzionare correttamente il file index1.jsp (che restituisce lo stesso risultato di index.jsp ma ottenuto mediante i tag jstl/xml), bisogna tuttavia ricordarsi, prima di tutto, di modificare i percorsi dei file XSL e XML definiti all'interno di essa (per ragioni di opportunità di sviluppo la porta di ascolto di Tomcat, in tali percorsi, è stata fissata in 8084).

```
'carichiamo il file XSL
objXSL.load(Server.MapPath("progetto.xsl"))

'la pagina Asp restituisce il risultato HTML della
                                        trasformazione
Response.Write(objXML.transformNode(objXSL))

%>
```

Inseriamo il file in una directory che vogliamo del nostro *web-server* (tipicamente Internet Information Server) e puntiamo un browser su di esso: come per magìa, l'output HTML della trasformazione comparirà all'interno della pagina.
Vediamo adesso di realizzare lo stesso lavoro con PHP, con particolare riferimento alla nuova versione 5 del linguaggio.
Per quanto riguarda PHP5, infatti, va detto che il nuovo supporto a XSLT, fornito dall'estensione XSL, va a sostituire la precedente implementazione in PHP4 che faceva riferimento alla libreria Sablotron. Sebbene la grande novità di PHP5 (in ambito XML) non sìa propriamente dovuta alla sua evoluzione nel supporto a XSLT (quanto piuttosto all'introduzione dell'estensione SimpleXml per agevolare l'accesso ai tag), è comunque importante sottolineare come questo cambiamento segni un vero e proprio salto di qualità, almeno rispetto alla precedente implementazione.
Ecco qui di seguito una semplice classe PHP5 che introduce le funzionalità XSLT nella pagina:

```
<?
class xml2xhtml {
    private $file_xml, $file_xsl, $risultato;

    function __construct($file_xml, $file_xsl) {
        // è il costruttore della classe, invocato
                                            automaticamente
        // su creazione di una nuova istanza

        $this->file_xml = $file_xml;
        $this->file_xsl = $file_xsl;

        $this->trasforma();
    }

    private function trasforma() {
        // questa funzione, privata, è quella
        // che effettua materialmente la trasformazione
        // E' accessibile soltanto dall'interno della
                                                   classe

        $doc = new DOMDocument();
        $xsl = new XSLTProcessor();

        $doc->load($this->file_xsl);
        $xsl->importStyleSheet($doc);

        $doc->load($this->file_xml);

        $this->risultato = $xsl-
                                    >transformToXML($doc);
        if (!$this->risultato) {
          $this->risultato = "E' impossibile effettuare la
                                  trasformazione richiesta";
        }
    }

    function render() {
        // questa funzione, accessibile dall'esterno
        // renderizza nella pagina il risultato della
        // trasformazione

        echo $this->risultato;
    }

}

// creiamo una nuova istanza dell'oggetto  xml2xhtml
$transformer = new xml2xhtml('progetto.xml',
                                            'progetto.xsl');
?>
```

Il file sopra riportato deve essere incluso nella pagina di destinazione della trasformazione. Per invocare quest'ultima, tuttavìa, occorre richiamare il metodo render(), che restituisce l'output HTML.
Vediamo adesso come eseguire lo stesso lavoro utilizzando il linguaggio Java.

COSA SONO I MODELLI DTD E GLI XML SCHEMA ?

La stesura di un documento XML impone in ogni caso il rispetto di alcune regole rigide, come la necessità di chiudere sempre i tag dopo la loro apertura, ecc... E' poi anche possibile, da parte dello sviluppatore, definire dei vincoli validi solo per il documento in questione, ai quali la struttura del file XML deve conformarsi.
La creazione di un modello DTD ("Document Type Definition") serve proprio a formalizzare questi vincoli, in modo da fornire un controllo sulla validità della struttura del file XML.
C'è da dire, tuttavìa, che il modello DTD soffre di alcune grosse limitazioni (non permette di stabilire, ad esempio, il numero esatto di occorrenze di un dato elemento, o il tipo di dato preciso a cui esso fa riferimento, ecc...). E' evidente, quindi, che tramite questo modello possiamo solo limitarci a definire in modo rapido e conciso un insieme di vincoli generici a cui è assoggettato il nostro file XML; se tuttavìa è necessario un maggiore controllo allora occorre utilizzare il XML Schema, che rappresenta un vero e proprio linguaggio di descrizione dei file XML.
Per approfondire maggiormente questi argomenti, consiglio di accedere ai seguenti link: http://www.w3schools.com/dtd e http://www.w3schools.com/schema .

Diciamo subito che per i nostri scopi potremmo utilizzare direttamente una *servlet*, che riceve in input gli URL dei file XML e XSL da processare e restituisce un risultato. Tuttavìa, così facendo, dovremmo effettuare un'inclusione di tale *servlet* all'interno di una pagina Jsp già esistente: noi, invece, sceglieremo una via alternativa, e decisamente più elegante. Utilizzeremo, infatti, un componente Javabeans, ossìa una classe dotata di alcune caratteristiche che fanno proprio al caso nostro:

```
package xslt.beans;

import javax.servlet.http.HttpServletRequest;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;
import java.net.*;

public class xsltJavabean implements Serializable {
  private Source xmlFile; //rappresenta il file xml
  private Templates xslFile; //rappresenta il file xsl
  private String xsltContext; // rappresenta il
                                      percorso dei file

  private TransformerFactory factory;
  // rappresenta il costruttore di trasformatori

  public xsltJavabean() {
    factory = TransformerFactory.newInstance();
  }

  public void setContextXslt(HttpServletRequest req)
                                                  {
    // solo nell'esempio di questo articolo, imposto
                                      esplicitamente
    // l'URL dei file xml e xsl; in un caso di
                                produzione, tuttavìa,
    // è evidente che i due percorsi possono essere
                                      rappresentati
    // da qualsiasi tipologìa di URL (anche, ad es.,
                                          file://...)

    // naturalmente, devo passare al bean l'oggetto
                                    request della jsp
    // come argomento del metodo in questione
    xsltContext = "http://"
                  + req.getServerName()
                  + ":" + req.getServerPort()
                  + req.getContextPath() + "/";
  }

  // permette di impostare l'URL del file XML
  public void setXmlURL(String url) throws
              MalformedURLException, IOException {
    xmlFile = openURL(url);
  }

  // permette di impostare l'URL del file XSL
  public void setXslURL(String url) throws
              MalformedURLException, IOException,
              TransformerConfigurationException {
    xslFile = factory.newTemplates(openURL(url));
  }

  private Source openURL(String url) throws
              MalformedURLException, IOException {
    URL u = new URL(xsltContext + url);
    return new StreamSource(u.openStream());
  }

  // effettua la trasformazione e ritorna il risultato
                                della trasformazione
  public String getOutput() throws
                              TransformerException {
    Transformer trans = xslFile.newTransformer();

    StringWriter risultato = new StringWriter();
    trans.transform(xmlFile, new
                              StreamResult(risultato));
    return risultato.toString();
  }
}
```

Ed ecco le righe principali della pagina Jsp che istanzia il suddetto bean (da notare l'uso dei tag Core, facenti parte delle librerie JSTL, grazie ai quali possiamo evitare di mischiare codice Java con i tag HTML, migliorando notevolmente la leggibilità anche da parte dei "non programmatori"):

```
<jsp:useBean class="xslt.beans.xsltJavabean"
                id="xslTrasforma" scope="request" />
<c:set value="${pageContext.request}"
  target="${xslTrasforma}" property="contextXslt" />
<c:set value="progetto.xml"
```

COSA SONO I COMPONENTI JAVABEANS ?

Nell'articolo abbiamo fatto riferimento ai JavaBean, per la realizzazione del motore XSLT. Questi componenti, in realtà, non sono altro che semplici classi Java, che rispettano però alcune caratteristiche di base:
hanno costruttore pubblico e privo di argomenti;
hanno variabili private e metodi pubblici getter e setter per ottenere l'accesso alle stesse;
implementano l'interfaccia Serializable (che tuttavìa è priva di metodi, serve soltanto al sistema per identificare il tipo corretto della classe).
I JavaBean sono utili soprattutto per "incapsulare" all'interno di componenti riusabili determinate logiche applicative: essi offrono poi anche altre funzionalità, come la possibilità di impostare uno scope, ovvero la visibilità del componente all'interno della web-application.
Per chi volesse approfondire autonomamente lo studio dei JavaBean, e più in generale di tutto ciò che concerne lo sviluppo orientato alla tecnologia Jsp, consiglio il seguente ottimo volume: JavaServer Pages di Hans BergSten (O' Reilly).

```
target="${xslTrasforma}" property="xmlURL" />
<c:set value="progetto.xsl"
    target="${xslTrasforma}" property="xslURL" />

<h2>Riepilogo delle attività:</h2>
<c:out value="${xslTrasforma.output}"
    escapeXml="false" />
```

Dicevamo, l'utilizzo di un componente Javabeans ci fornisce una serie di agevolazioni: intanto, grazie ai suoi metodi *setter* e *getter* possiamo creare facilmente proprietà di sola lettura (come *output*, definita dal metodo getOutput) o di sola scrittura (come *xmlURL* e *xslURL*, che derivano rispettivamente dai metodi setXmlURL e setXslURL). Il componente può essere istanziato nella pagina in modo molto semplice, tramite l'azione <jsp:useBean ... />, e con la stessa facilità possiamo anche impostarne le varie proprietà (tramite ad esempio i tag Core <c:set ...>) o richiedere il valore delle stesse (come nel caso di *output*). Naturalmente i punti di forza dei componenti Javabeans non si fermano qui (e questo giustifica in pieno la grande diffusione di cui essi godono nell'ambito dello sviluppo *Java-oriented*), anche se una trattazione approfondita dell'argomento andrebbe di sicuro affrontata in ben altra sede.
Anche la scelta di utilizzare i tag JSTL ci torna utile, al fine di evitare l'inserimento di codice Java direttamente nella pagina Jsp. A questo proposito, è bene ricordare che JSTL, oltre ad annoverare la libreria Core (contenente tutte le varie funzioni di utilità), include anche altri insiemi di tag, tra i quali quelli legati a Xml, in grado da soli di erogare funzionalità XSLT all'interno della pagina.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib
    uri="http://java.sun.com/jsp/jstl/core"
    prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/xml"
    prefix="x"%>

<c:import var="xml_filename"
 url="http://localhost:8084/XSLT_Server/progetto.
xml
" />
<c:import var="xsl_filename"
 url="http://localhost:8084/XSLT_Server/progetto.
xsl"
/>

<h2>Riepilogo delle attività:</h2>
<x:transform xml="${xml_filename}"
    xslt="${xsl_filename}" />
```

Le poche righe di codice sopra riportate (tra cui spiccano, naturalmente, l'importazione della libreria di tag jstl/xml e il suo utilizzo mediante l'azione x:transform), sono infatti in grado, da sole, di operare la trasformazione XSLT realizzata in precedenza tramite il componente Javabeans.

CONCLUSIONI

Nel presente articolo abbiamo analizzato un semplice caso pratico di trasformazione XSLT lato-server. Non è difficile, a questo punto, comprendere il grande potenziale racchiuso in questa tecnologìa: grazie ad essa, infatti, possiamo ottenere il massimo livello di separazione tra le logiche applicative e quelle rappresentative, con la garanzia, peraltro, che il prodotto finale sarà fruibile da chiunque allo stesso modo, indipendentemente dai requisiti *software* della piattaforma di destinazione. Tutte caratteristiche che aumentano a dismisura la libertà di scelta del programmatore.
C'è anche da dire, infine, che ormai tutti i linguaggi moderni supportano nativamente funzionalità avanzate legate a XML, quando addirittura non basano totalmente la loro struttura "grammaticale" su questo formato (mi riferisco ai vari XUL di Mozilla, XAML di Microsoft, ecc...).
Una ragione in più, quindi, per accrescere il più possibile la propria conoscenza su questi strumenti, che (c'è da scommetterci) giocheranno un ruolo da protagonisti assoluti nel futuro mondo della programmazione.

Enrico Viale

L'AUTORE

Enrico Viale è specializzato nello sviluppo di applicazioni sia web-oriented che desktop. Chi desidera contattarlo per chiarimenti riguardo all'articolo, o per qualsiasi altro motivo, può farlo all'indirizzo enrico.viale@yahoo.it .

COS'È SIMPLEXML ?

Nel testo si è fatto riferimento a SimpleXml, la vera grande novità introdotta da PHP5 nell'ambito di XML. Questa estensione offre un accesso notevolmente semplificato alle varie funzionalità di lettura e scrittura di documenti XML. Essa, in particolare, tratta la struttura di un file XML come un insieme di oggetti generati in modo automatico e chiamati con il nome dei nodi a cui si riferiscono: questo approccio consente di risparmiare una grande quantità di codice rispetto invece al metodo tradizionale basato sul DOM. Ecco un semplice esempio di utilizzo di SimpleXml:

```
$xml =
    simplexml_load_string($xmlstr);

/* For each <movie> node, we echo
    a separate <plot>. */
foreach ($xml->movie as $movie) {
    echo $movie->plot, '<br
    />';
}
```

Maggiori informazioni (e ulteriori esempi) sull'uso di SimpleXml si possono reperire direttamente al seguente sito web: http://it2.php.net/simplexml .

EFFETTI SPECIALI PER LE APPLICAZIONI

IMPARIAMO COME UTILIZZARE LE API DI SWING-BUG PER REALIZZARE APPLICAZIONI DESKTOP DOTATE DI INTERFACCE GRAFICHE AVANZATE E DOTATE DI ANIMAZIONI DECISAMENTE INSOLITE PER SOFTWARE SVILUPPATO IN JAVA

Lo sviluppo di applicazioni grafiche 2D in Java non è mai stato semplice. In questo articolo introdurremu la libreria Swing-Bug, nata all'interno di un blog (http://www.blogof bug.com/) e poi divenuta un progetto vero e proprio (https://swing-bug.dev.java.net/). Che ci aiuterà a sviluppare interfacce grafiche accattivanti eliminando la complessità della tecnologia nativa.

JCAROUSELMENU

La componente *JCarouselMenu* è un estensione di *javax.swing.JPanel* che permette la creazione di un originalissimo tipo di menù, suddiviso in due parti: quella destra comprendente la lista di voci testuali componenti il menù e quella sinistra contenente l'immagine associata ad ogni voce. L'azione di cambio di selezione produce un effetto grafico che fa ruotare le immagini sulla sinistra fino a quella relativa all'elemento cliccato. In Figura 1 è visualizzato uno screenshot della nostra applicazione di esempio che fa uso del *JCarouselMenu*.

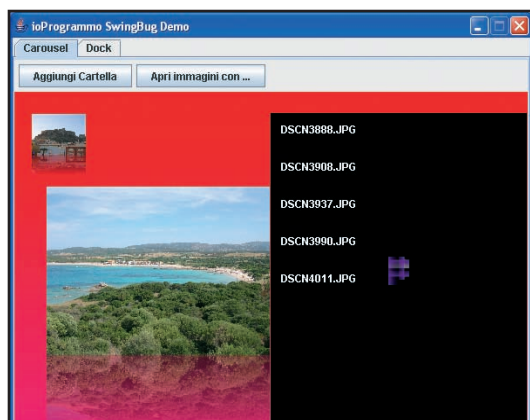



***Figura 1:** Visualizziamo le immagini con l'aiuto del JCarouselMenu*

La nostra implementazione presenta due bottoni nella parte superiore tramite i quali è possibile specificare una cartella contenente immagini ed un programma per la visualizzazione esterna delle immagini stesse. Nella parte centrale è posizionato un oggetto di tipo *JCarouselMenu* creato nel seguente modo:

```
JCarouselMenu carousel = new JCarouselMenu();
carousel.setBackground(Color.RED, new Color(255, 0,
                                              122));
```

Come si può notare il background è costituito da due elementi in modo da creare un gradiente prodotto da due oggetti di tipo *java.awt. Color*. L'azione legata al bottone "Aggiungi Cartella" invoca il seguente metodo per popolare il menù:

```
public void addFolder(File folder) {
    File[] images = folder.listFiles(newImageFilter());
    for (File f : images) {
      if (!f.isDirectory()) {
        try {
          this.carousel.add(new
              ImageAction(f.getName(), f.toURL(),
                          f.getAbsolutePath()));
        } catch (MalformedURLException ex) {
          JOptionPane.showMessageDialog(this,
  ex.getMessage(), "Impossibile aggiungere l'immagine
      " + f.getName(), JOptionPane.ERROR_MESSAGE);
Logger.getLogger("global").log(Level.SEVERE, null, ex);
        }
      }
    }
    this.repaint();
  }
```

Il metodo itera su tutti i file di tipo immagine e, per ciascuno di essi crea e aggiunge un oggetto di tipo *ImageAction* al *JCarouselMenu*.

```
public class ImageAction extends
                  AbstractCarouselMenuAction {
    private String imagePath;
    public ImageAction(String label, URL image,
                        String imagePath){
      super(image,label);
      this.imagePath = imagePath;
```

```
    }
    public void actionPerformed(ActionEvent
                                    actionEvent) {
System.out.println(actionEvent.getSource());
        try {
            java.lang.Runtime.getRuntime().exec("\""
                + command + "\" \"" + imagePath+"\"");
        } catch (IOException ex) {
Logger.getLogger("global").log(Level.SEVERE,
                "Impossibile eseguire il comando "
                    + command + " " + imagePath,
                ex);
        }
    }
}
```

Il metodo *actionPerformed* viene invocato ad ogni doppio click del mouse sulla relativa voce di menù ed invoca il tool esterno per la visualizzazione dell'immagine. È possibile cambiare l'applicazione esterna tramite il bottone "Apri immagini con ...", il default è *explorer*.

LAYEREDDOCKPANEL

Nel gergo informatico il *dock* è una barra dove si trovano i collegamenti ad una serie di programmi in attesa di essere eseguiti e la sua implementazione più conosciuta è quella residente nell'interfaccia grafica del sistema operativo MAC OS X, dove le icone dei programmi sono dei componenti attivi che si ingrandiscono al passaggio del mouse per evidenziare la possibilità di eseguire la relativa applicazione. In Figura 2 è visualizzata l'implementazione di tale componente fornita da Swing-Bug.
La classe che lo implementa è *LayeredDockPanel*, un'estensione di *JPanel* progettata per essere inserita all'interno di un componente di tipo *Frame*, in rilievo confronto agli altri elementi presenti nel *container*. Cliccando sul bottone in alto a sinistra viene

***Figura 2:** Il componente dock contenente le nostre applicazioni*

mostrato un *dialog* che consente di specificare una nuova applicazione da aggiungere al *dock* mediante il seguente metodo:

```
public void addApplication(String name, final String
                              command, String image) {
    ImageLabel imgLabel = new ImageLabel(new
                              ImageIcon(image),32,32);
    imgLabel.addMouseListener(newMouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            if (e.getClickCount() >= 2) {
                try {
java.lang.Runtime.getRuntime().exec("\"" +
command + "\"");
                } catch (IOException ex) {
Logger.getLogger("global").log(Level.SEVERE,
    "Impossibile eseguire il comando " + command, ex);
                }
            }
        }
    });
    this.dockPanel.addDockElement(imgLabel,name);
    this.repaint();
}
```

Il metodo riceve nome, comando e icona dell'applicazione, crea un oggetto di tipo *ImageLabel* e lo aggiunge al *dock* invocando il metodo *addDockElement*. Inoltre, aggiunge ad ogni icona un *listener* responsabile di eseguire il comando legato all'applicazione.

COME INIZIARE

- **É indispensabile avere installato sul computer Java 2 Standard Edition 1.5 o superiore. É preferibile utilizzare l'ultima versione disponibile dal sito http://java.sun.com.**
- **Attualmente non è stata rilasciata una versione binaria (JAR file) della libreria. È comunque possibile scaricare i sorgenti tramite CVS seguendo le istruzioni presenti sul sito del progetto (https://swing-bug.dev.java.net/).**
- **Il file swing-bug.zip contiene sia l'applicazione di esempio presentata nell'articolo sia il codice e la documentazione della libreria Swing-Bug. Entrambi i progetti sono stati creati con Netbeans con cui si possono facilmente importare.**
Per testare l'applicazione di esempio posizionarsi nella cartella dist ed eseguire il comando "java -jar SwingBugSample.jar".

JBOOKPANEL

In questa sezione introdurremo una delle ultime novità introdotte nel progetto Swing-Bug: il *JBookPanel*. Si tratta di un contenitore di oggetti di tipo *javax.swing.JComponent* visualizzabili con l'effetto libro, ovvero è possibile passare da un componente all'altro semplicemente "sfogliando" la pagina con il mouse, come visualizzato in **Figura 3**.
Per la nostra applicazione di esempio ci siamo limitati a visualizzare alcune immagini interne al nostro JAR file:

```
JComponent[] pages = new JComponent[4];
```

```
pages[0] = new JLabel(new
    ImageIcon(this.getClass().getResource("/it/ioprogram
        mo/example/images/foto1.jpg")));
pages[1] = new JLabel(new
    ImageIcon(this.getClass().getResource("/it/ioprogram
        mo/example/images/foto2.jpg")));
pages[2] = new JLabel(new
    ImageIcon(this.getClass().getResource("/it/ioprogram
        mo/example/images/foto3.jpg")));
pages[3] = new JLabel(new
    ImageIcon(this.getClass().getResource("/it/ioprogram
        mo/example/images/foto4.jpg")));

JBookPanel bookPanel = new JBookPanel();
bookPanel.setPages(pages, 262, 350);
    bookPanel.setMargins(30, 40);
bookPanel.setBackground(new Color(157,185,235));
bookPanel.setSoftClipping(true);
bookPanel.setBorderLinesVisible(true);
```

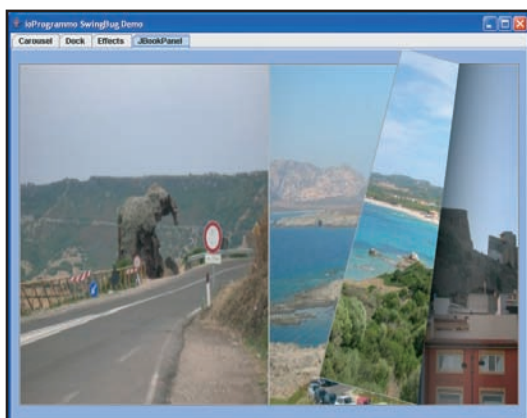



Figura 3: JBookPanel utilizzato come album fotografico

EFFECTS

In questo paragrafo vedremo come sia possibile inserire semplici effetti animati utilizzando i metodi messi a disposizione delle API di Swing-Bug. Attualmente, la libreria contiene 18 differenti effetti, alcuni utilizzabili singolarmente altri annidabili tra di loro, che implementano l'interfaccia Effect e che si possono applicare unicamente su oggetti di tipo EffectsPanel: una specifica estensione di JPanel.
In **Figura 4** è visualizzata l'applicazione di esempio contenente una serie di effetti grafici.
Nella parte sinistra è stato posizionato un oggetto di tipo *javax.swing.JList*, a cui è stato aggiunto un *listener* che si occupa di scatenare un effetto di tipo *ListCellBurst* al cambio di selezione. Questo effetto produce "un'esplosione" del testo contenuto nell'elemento selezionato che si propaga all'interno della finestra.

```
public void valueChanged(ListSelectionEvent e) {
```

```
        if (!e.getValueIsAdjusting()){
            if (reflectBox.isSelected()) {
                EffectsContainer.this.addEffect(new
                    ReflectEffect(new ListCellBurst(list,
            list.getSelectedIndex()), EffectsContainer.this));
            } else {
                EffectsContainer.this.addEffect(new
                ListCellBurst(list, list.getSelectedIndex()));
            }
        }
}
```

Notare che, nel caso in cui la checkbox "Reflect" sia selezionata l'effetto *ListCellBurst* viene inglobato all'interno di un oggetto *ReflectEffect* responsabile di creare un effetto specchio. Inoltre, nella parte centrale dello schermo è stato utilizzato un effetto di tipo *ParticleEffect* che, sul movimento del mouse, produce una reazione simile a quella delle particelle

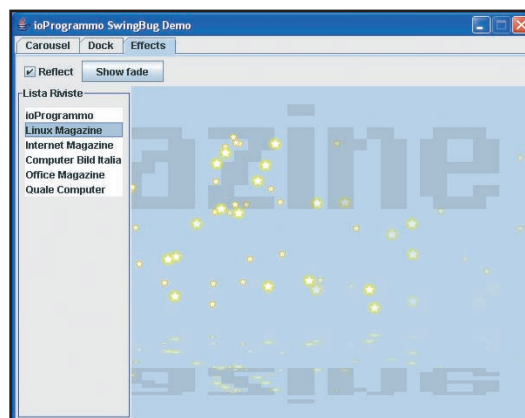



Figura 4: Una dimostrazione di alcuni effetti grafici

su cui incide la forza di gravità.

```
public void mouseMoved(Point position) {
    Effect theEffect = null;
    if (Math.random() > 0.5) {
        theEffect = new ParticleEffect(starSmall,
            (int) position.getX(), (int) position.getY(), 40,
                (this.getHeight() / 3) * 2);
    } else {
        theEffect = new ParticleEffect(star, (int)
            position.getX(), (int) position.getY(), 40,
                (this.getHeight() / 3) * 2);
    }

    if (this.reflectBox.isSelected()) {
        this.addEffect(new ReflectEffect(theEffect,
                                    this));
    } else {
        this.addEffect(theEffect);
    }
}
```

Fabrizio Fortino

GESTIONE DEI DATI CON I FRAME

IL COLLEGAMENTO XMLHTTPREQUEST NON È IL SOLO MODO DI REPERIRE I DATI DA ELABORARE IN JAVASCRIPT, VEDREMO ALL'OPERA UNA TECNICA CHE CI CONSENTE DI FARE A MENO DEL LATO SERVER E DI PROGETTARE APPLICAZIONI OFF-LINE

Più volte abbiamo scritto sulle modalità di recupero dei dati dal server in applicazioni Web di tipo AJAX : si effettua una chiamata XMLHttpRequest, il server elabora una risposta utilizzando un linguaggio di programmazione come PHP, ASP, Java ecc... e restituisce una stringa con i dati (in formato XML, JSON o altro) che il client elaborerà secondo il flusso del programma.
Questa volta però vedremo un'altra tecnica di recupero dati, la Hidden Frame, che ci consentirà addirittura di fare a meno del tutto di pagine che utilizzino un linguaggio di programmazione lato server!

IL PRINCIPIO DI BASE HIDDEN FRAME

La tecnica della Hidden Frame si basa appunto in una <FRAME> o <IFRAME> nascosta dove verranno caricati i dati in forma di Array od oggetto Javascript.
La cosa è più semplice a comprendere vedendo un semplice esempio.
Poniamo di avere scritto, in un file che chiameremo data.js, i dati di un anagrafica ad esempio come Array di Array:

```
var anagrafica = [
['Mario','Rossi','Via Verdi 10',' Milano'],
['Antonio','Bianchi','Via Leopardi 1','Genova']
];
```

Adesso creeremo una pagina web, che chiameremo data.html, che serve solo come "contenitore" del file di dati che verrà incluso come risorsa Javascript esterna:

```
<html>
<head>
        <script src="data.js"
                type="text/javascript"></script>
</head>
        <body></body>
</html>
```

Aggiungeremo a questo punto, al codice html, del codice Javascript che, dopo il caricamento della pagina, eseguirà una funzione:

```
<script language="javascript"
                type="text/javascript">
        window.onload = function () {
            try{
                        top.loadEnd(window);
                }
                catch(e){}
        }
</script>
```

Notate però che la funzione loadEnd non viene richiamata nella pagina stessa, ma nella pagina superiore (*top*). Questo perché la pagina *data.html*, che contiene l'Array di dati, è destinata ad essere inserita in una <IFRAME> di una pagina contenitore (referenziata appunto dall'oggetto top). Una cosa che a volte ci dimentichiamo è infatti che le varie finestre o frames in Javascript possono comunicare (a condizione di essere tutte provenienti dallo stesso sito), è insomma possibile da una finestra richiamare funzioni o utilizzare variabili contenute in un'altra finestra. Questa particolarità nel nostro caso ci torna molto utile: poniamo infatti il caso che la pagina data.html (che a sua volta include il file *data.js*) abbia, considerato l'Array, una dimensione di diverse centinaia di Kb, la pagina contenitore mostrerà allora un messaggio di caricamento dati finché *data.html* non sarà stata del tutto caricata, a quel punto dalla stessa data.html verrà richiamata la funzione *loadEnd* della pagina contenitore ed i dati potranno cominciare ad essere utilizzati.
Vediamo quindi, sinteticamente, come viene strutturata la pagina che contiene l'<IFRAME> (che poi sarà quella che contiene la

nostra applicazione):

```
<html>
<head>
...
</head>
      <div id="wait">
            Attendere, caricamento dati in
                                        corso ...
      </div>
      <iframe id="dataFrame" width="0"
   height="0" src="data.html" frameborder="0">
</iframe>
<div id="main" style="display:none">
      Interfaccia applicazione
      </div>
</html>
```

Abbiamo cioè due <DIV> di cui il secondo è nascosto tramite l'attributo CSS display e una <IFRAME> di dimensioni e bordo a 0, quindi invisibile, che carica *data.html*, la pagina contenente i dati. Nello stato iniziale sarà quindi visibile solo il <DIV> contenente il messaggio di attesa. Inseriamo allora la funzione *loadEnd* che verrà richiamata dall' <IFRAME> al termine del caricamento :

```
<script language="javascript"
                        type="text/javascript">
      function loadEnd (srcWindow){
            ...
      }
</script>
```

Nel corpo della funzione scriviamo prima il codice necessario a nascondere il <DIV> con il messaggio di attesa e a mostrare l'altro <DIV>:

```
      ...
var div1 = document.getElementById("wait");
      var div2 =
            document.getElementById("main");
      div1.style.display = "none";
      div2.style.display = "";
      ...
```

poi recuperiamo i dati rappresentati dalla variabile *anagrafica* dichiarata a livello di <IFRAME> e li assegniamo a una variabile locale dallo stesso nome (ricordiamoci che la finestra di <IFRAME> è qui rappresentata dall'argomento *srcWindow* della funzione *loadEnd*):

```
      var anagrafica = srcWindow.anagrafica;
```

Quindi possiamo utilizzare i dati, ad esempio per creare una lista di nomi:

```
      var s = "<h3>DATI:</h3>";
      for(var i=0;i<anagrafica.length;i++){
            var rowPersona = anagrafica[i];
            s += "<li>" +
         rowPersona.join(" ") + "</li>";
      }
      div2.innerHTML = s;
```

Eseguendo la pagina in un browser avremo quindi dapprima il messaggio di attesa di cui in figura 1, quindi il risultato dell'utilizzo dei dati di cui alla figura 2.

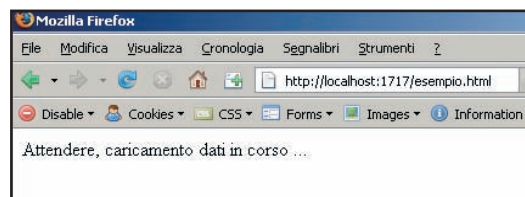



Fig. 1: ***Messaggio di attesa***

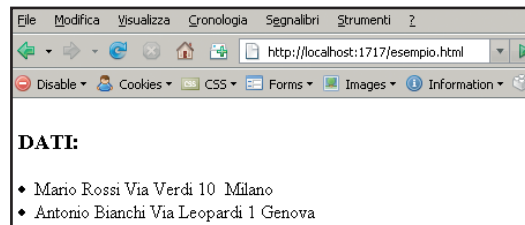



Fig. 2: ***Caricamento completato***

UTILITÀ DELLA TECNICA

Ma quand'è che questa tecnica di storage e recupero dei dati può essere utile?
La tecnica dell'Hidden Frame rappresenta una risorsa preziosa in alcune condizioni:

- si ha a che fare con una (relativamente) limitata quantità di dati (es. catalogo prodotti, foto gallery, listini prezzi ecc...)
- i dati cambiano poco spesso (così da beneficiare del *caching* del browser)
- si vuol rendere indipendente l'applicazione da un linguaggio lato server oppure si prevede di distribuire l'applicazione anche off-line

DAL DB ALL'ARRAY JAVASCRIPT

C'è però un problema che alcuni si saranno a questo punto posti : se i dati sono contenuti in

ANCHE CON XML

Un altro modo per ottenere un Array Javascript da una tabella è esportarla in formato XML e trasformarla mediante uno stylesheet XSLT. Nel codice allegato è riportato un esempio sia del file di dati XML prodotto da un'esportazione da Access 2007 che dal file XSL che può essere usato per produrre l'array.

origine in un database come fare a trasformarli in un array Javascript?
Certo si potrebbe anche scrivere una piccola utility di trasformazione da una tabella di DB ad un array, ma c'è anche un altro "trucchetto" abbastanza ingegnoso:

1. esportiamo la tabella in un foglio di Excel eliminando la prima riga se contiene l'intestazione
2. mettiamo che i dati siano nelle prime tre colonne del foglio (A,B e C) ci portiamo nella colonna D della prima riga contenente i dati e scriviamo una funzione di concatenazione di stringhe come:

```
="['" & A1 & "','" & B1 & "','" & C1 & "'],"
```

Il risultato sarà l'array che rappresenta la riga,ad esempio:

```
['Mario Rossi','Via Verdi 10',' Milano'],
```

3. copiamo la funzione nelle celle successive della colonna D ottenendo così, nella colonna D, gli Array corrispondenti alle righe
4. a questo punto basta copiare il contenuto delle celle della colonna D in un editor di testo:

```
...
['Mario','Rossi','Via Verdi 10',' Milano'],
['Antonio','Bianchi','Via Leopardi 1','Genova'],
```

eliminare l'ultima virgola:

```
...
['Mario','Rossi','Via Verdi 10',' Milano'],
['Antonio','Bianchi','Via Leopardi 1','Genova']
```

e racchiudere il tutto tra le parentesi quadre assegnando l'array ad una variabile:

```
var myArray = [
['Mario','Rossi','Via Verdi 10',' Milano'],
['Antonio','Bianchi','Via Leopardi 1','Genova']
];
```

Attenzione però, poiché la destinazione deve essere un codice valido in Javascript, e noi abbiamo usato il carattere di virgoletta singola come separatore, la funzione Excel dovrà prevedere il replace del carattere di virgoletta singola " ' " eventualmente presente nel testo con l'*escape* " \' ", la funzione completa sarebbe quindi :

```
="['" & SOSTITUISCI(A1;"'";"\'") & "','" &
SOSTITUISCI(B1;"'";"\'") & "','" &
SOSTITUISCI(C1;"'";"\'") & "'],"
```

LA PROVA SU STRADA: CALCOLO DEL CODICE FISCALE

È arrivato quindi il momento di applicare la tecnica dell'Hidden Frame ad un caso reale, per questo scegliamo il calcolo del codice fiscale, così, oltre che prendere confidenza con la tecnica, ci ritroveremmo ad avere anche una piccola applicazione di una certa utilità pratica che può essere usata anche off-line.
L'algoritmo di calcolo del codice fiscale non è difficile (dopo lo vedremo) il problema è che richiede un codice corrispondente al comune di nascita della persona (es. per Milano F205, Roma H501 e così via) quindi, tra comuni e stati esteri, abbiamo una tabella (e il conseguente array) di oltre 8000 righe.
L'esportazione della tabella dei codici comuni in un array Javascript produce un file di 248 Kb: si tratta di un caso un po' limite per usare la tecnica dell'Hidden Frame (ricordiamoci che il client scarica comunque tutti questi dati), comunque ancora può andare...
L'applicazione consisterà in una form dove ci sono delle textbox per nome e cognome, una select M/F per il sesso, un controllo DateBox per la data di nascita e un controllo Autosuggest per il comune di nascita.

L'INTERFACCIA UTENTE

Parlando di "controlli" DateBox e Autosuggest non mi riferisco naturalmente a elementi HTML particolari, ma ad elementi standard HTML (Input, Select ecc...) per cui del codice Javascript definisce particolari comportamenti.
Un esempio classico è l'Autosuggest che non è altro che un normale elemento <INPUT> al quale viene associata un'azione sull'evento *onkeyup* che consiste nel mostrare una lista di suggerimenti in un <DIV> posizionato al di sotto dell'elemento. Ebbene, quello che si definisce "controllo" è l'insieme di codice Javascript, CSS e HTML necessario a far comportare uno o più elementi HTML in un certo modo.
Nel tentativo di dare organicità a quell'insieme di tecnologie che comprendono lo sviluppo web su lato client, dette anche AJAX, WEB

2.0 ecc..., stanno fiorendo numerosi c.d. frameworks Javascript (Yahoo! User Interface Library, Rico, Qooxdoo, Dojo ecc...).
Lo scopo di tali frameworks e spesso quello di offrire al programmatore un abstraction layer che consenta di non curarsi delle differenze dei vari browser, e mettere a disposizione dei controlli già pronti che mimano quelli che più spesso si trovano nei sistemi operativi (tabs, tree-view, dateBox, autocomplete ecc...).
Non nego che a volte la tentazione di ricorrere ad un framework che offre un'abbondante scelta di GUI (Graphic User Interface) già pronta può essere forte, tuttavia c'è un rovescio della medaglia che alcune volte non si considera adeguatamente: Talvolta la dimensione del framework è superiore a quella del progetto, ad esempio: utilizzando come controllo Autosuggest quello presente nella YUL (Yahoo! User Interface Library) la dimensione complessiva degli script da includere nella pagina sarebbe stata di 183 Kb, mentre utilizzando un Autosuggest "fatto in casa" tutto il codice sta in un file di 12 Kb! Circa 15 volte meno!
A questo punto è evidente che la nostra scelta sarà quella di utilizzare, per DateBox, e Autosuggest, due script che ci siamo autocostruiti. Nel progetto allegato al CD troverete i due scripts completi, quello che a noi interessa, in questa fase, è vederne l'utilizzo pratico.
Entrambi si basano sull'aggiunta di funzionalità a elementi esistenti nel codice HTML.
Per il DateBox avremo un elemento che contiene tre <INPUT> che rappresentano giorno,mese ed anno, ad esempio :

```
<td id="dtNasc">
<input type="text" maxlength="2"
          class="datePart" id="day" name="day">
/
<input type="text" maxlength="2"
      class="datePart"id="month" name="month">
/
<input type="text" maxlength="4"
         class="datePart" id="year" name="year">
</td>
```

La definizione di questo complesso di elementi come DateBox e l'aggiunta delle funzionalità corrispondenti avviene, sul versante del codice Javascript, attraverso l'istanza di un nuovo oggetto:

```
var oContainer =
              document.getElementById("dtNasc");
dataNasc = new DateBox(oContainer);
```

Fig. 3: Controllo DateBox con valore errato

Fig. 4: Controllo DateBox con valore esatto

Le funzionalità che vengono aggiunte sono:

- Controllo correttezza nella digitazione e segnalazione visiva (il contenuto delle textbox apparirà in rosso se i tre valori non sono una data valida, verde se lo sono).
- Proprietà value dell'oggetto DateBox che rappresenta un oggetto di tipo Date o null.

Abbastanza semplice è anche l'utilizzo di Autosuggest dove il codice HTML su cui si applica il controllo è:

```
<input type="text" id="luogoNasc"/>
```

e il costruttore Javascript sarà:

```
codComuneAutocomplete = new
                          AutoSuggestControl(
document.getElementById("luogoNasc"),
                    new CodeSuggestion(),
                    {width:400}
                    );
```

i tre argomenti forniti sono:

1. Riferimento all'elemento <INPUT> su cui si installa l'Autosuggest
2. Un oggetto che deve contenere il metodo getArray che deve restituire l'array di stringhe da mostrare nel box sottostante dato

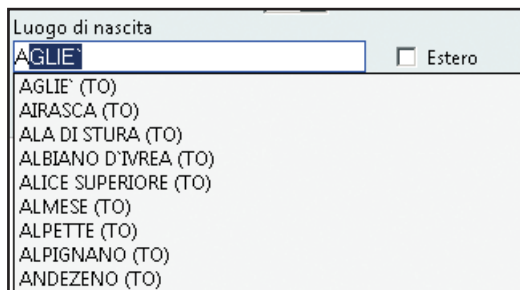



Fig. 5: Controllo Autosuggest in azione

un valore di confronto passato come argomento, nel nostro caso ad esempio la struttura del metodo sarà:

```
CodeSuggestion.prototype.getArray =
function  (searchValue){
var aSuggestions = [];

... //composizione dell'array
return aSuggestions;
}
```

3. Il terzo argomento è invece un oggetto contenente le opzioni di configurazione (attualmente limitate alla sola proprietà width ovvero la larghezza del box dove appaiono i valori, ma configurata come oggetto in modo da garantire una maggiore flessibilità in caso di modifiche future)

Alla fine l'interfaccia utente sarà quella mostrata in figura 6

Fig. 6: l'interfaccia utente

con una <IFRAME> nascosta dove vengono caricati i dati ed uno *splash* screen iniziale di attesa caricamento che possiamo vedere in figura 7

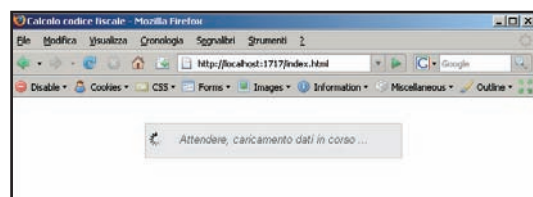



Fig. 7: Splash screen iniziale

CALCOLO DEL CODICE FISCALE

Vediamo adesso l'algoritmo vero e proprio di calcolo del codice fiscale.

Cognome (3 lettere) - si prendono le prime tre consonanti del cognome (ad esempio SML per SMELZO), se le consonanti sono insufficienti si prelevano anche le vocali, sempre nel loro ordine (ad esempio RAA per ARA); se anche le vocali sono insufficienti si usano le X (ad esempio BOX per BO).

Nome (3 lettere) – vengono prese la prima, la seconda e la quarta consonante del nome (ad esempio FNC per FRANCESCO), se le consonanti sono meno di quattro le tre, se non sono sufficienti si prendono anche le vocali in maniera analoga a quanto visto con il cognome.

Anno di nascita (2 cifre) – si prendono le 2 ultime cifre dell'anno di nascita

Mese di nascita (1 lettera) – si trasforma il mese di nascita in lettera secondo la **tabella**:

Lettera	Mese	Lettera	Mese
A	gennaio	L	luglio
B	febbraio	M	agosto
C	marzo	P	settembre
D	aprile	R	ottobre
E	maggio	S	novembre
H	giugno	T	dicembre

Giorno di nascita (2 cifre) – si prendono le cifre del giorno di nascita (se è un numero fino a 9 si antepone uno 0), per le donne si somma a tale numero 40.

Comune di nascita (4 caratteri alfanumerici) – si prende il codice corrispondente al comune (o allo stato estero) di nascita composto da una lettera e 3 cifre numeriche.

Codice di controllo (1 lettera) - Partendo dai 15 caratteri ricavati in precedenza, si calcola il codice di controllo: da una parte si mettono i caratteri alfanumerici che si trovano in posizione dispari (il 1°, il 3°, ecc.) e da un'altra quelli che si trovano in posizione pari (il 2°, il 4°, ecc.). Questi caratteri vengono convertiti in numeri utilizzando le seguenti tabelle:

A questo punto tutti i valori vengono somma-

Carattere	Valore	Carattere	Valore	Carattere	Valore
0	1	C	5	O	11
1	0	D	7	P	3
2	5	E	9	Q	6
3	7	F	13	R	8
4	9	G	15	S	12
5	13	H	17	T	14
6	15	I	19	U	16
7	17	J	21	V	10
8	19	K	2	W	22
9	21	L	4	X	25
A	1	M	18	Y	24
B	0	N	20	Z	23

CARATTERI ALFANUMERICI DISPARI

NOTA

ALGORITMO DI CALCOLO DEL CODICE FISCALE

All'indirizzo http://it.wikipedia.org/wiki/Codice_fiscale troviamo una spiegazione dettagliata ed esauriente dell'algoritmo di calcolo del codice fiscale.

Carattere	Valore	Carattere	Valore	Carattere	Valore
0	0	C	2	O	14
1	1	D	3	P	15
2	2	E	4	Q	16
3	3	F	5	R	17
4	4	G	6	S	18
5	5	H	7	T	19
6	6	I	8	U	20
7	7	J	9	V	21
8	8	K	10	W	22
9	9	L	11	X	23
A	0	M	12	Y	24
B	1	N	13	Z	25

CARATTERI ALFANUMERICI PARI

ti e divisi per 26, il resto della divisione verrà convertito in carattere utilizzando questa tabella:

Resto	Lettera	Resto	Lettera	Resto	Lettera
0	A	9	J	18	S
1	B	10	K	19	T
2	C	11	L	20	U
3	D	12	M	21	V
4	E	13	N	22	W
5	F	14	O	23	X
6	G	15	P	24	Y
7	H	16	Q	25	Z
8	I	17	R		

RESTO

IMPLEMENTAZIONE DELL'ALGORITMO IN JAVASCRIPT

Per implementare il calcolo del codice fiscale in Javascript abbiamo realizzato un file separato (presente nel codice allegato con il nome *codice Fiscale.js*) dove per prima cosa andremo a estendere alcuni oggetti base come String e Date con funzioni che successivamente ci saranno utili; riportiamo le intestazioni delle funzioni di estensione :

```
String.prototype.isVocal = function (){
//determina se il primo carattere della stringa è
                                        una vocale
// es. "a".isVocal() [true]
}
String.prototype.toCharArray = function (){
//restituisce l'array di caratteri che compongono
                                        la stringa
}
String.prototype.extractLetters = function
                                (typeLetter,casing){
/*
estrae tutte le lettere dalla stringa come array di
                                caratteri escludendo
i caratteri che non sono lettere come backspace,
                                   tab, spazi ecc...
typeLetter può essere "v" o "c" per estrarre solo
                               vocali o solo consonanti
casing può essere "u" o "l" per ottenere lettere
                        tutte maiuscole o tutte minuscole
*/
}
String.prototype.padLeft = function
                                   (padChar,padLen){
/*
aggiunge dei caratteri di riempimento a sinistra
          della stringa, ad esempio "1" diventa "01"
padChar è il carattere da usare come riempimento
padLen è la lunghezza complessiva che dovrà
           avere la stringa compresi i caratteri di
                                         riempimento
*/
}
```

Definiamo quindi il prototipo dell'oggetto, che racchiuderà tutte le funzionalità di calcolo, che chiameremo CodiceFiscale:

```
function CodiceFiscale
    (nome,cognome,sesso,codComune,dataNasc){
        this.nome = nome || '';
        this.cognome = cognome || '';
        this.sesso = ((sesso || 'M')=='M')?'M':'F';
        this.codComune = codComune;
        this.dataNasc = dataNasc;
}
```

Com'è facile intuire nel costruttore vengono passati anche i dati occorrenti al calcolo. Per controllare meglio i dati di input prevederemo anche un metodo di creazione dell'istanza che effettua anche i controlli sui dati e solleva eventuali eccezioni:

```
CodiceFiscale.createInstance = function
    (nome,cognome,sesso,codComune,dataNasc){
/*
N.B. Le varie funzioni di controllo come
```

NOTA SUGLI ESEMPI PRESENTI NELL'ARTICOLO

**Negli esempi di codice che troverete nell'articolo vengono usate alcune funzioni come isNull, isNulString,$(), l'oggetto $E ecc...
Si tratta di funzioni ed oggetti definiti in una piccola libreria che l'autore ha sviluppato (una specie di Prototype, ma limitato all'essenziale) il cui scopo è fornire alcuni metodi di utilità (come isNull ecc...) non presenti in Javascript oppure delle scorciatoie sintattiche (come $("element") invece di document.getElementById("element")).
Questa libreria, chiamata Core.js, è allegata al codice d'esempio**

```
isNullString ecc... sono definite altrove
*/
var missingData = 'Manca dato:{0}';
var badData = 'Dato non conforme:{0}';
if (isNullString(nome)) throwError(1,
    missingData.printf('Nome'));
if (isNullString(cognome)) throwError(1,
    missingData.printf('Cognome'));
if (isNullString(codComune))
    throwError(1, missingData.printf('Codice
    Comune'));
if (isNullString(sesso)) throwError(1,
    missingData.printf('Sesso'));
if (isNull(dataNasc)) throwError(1,
    missingData.printf('Data di nascita'));
if (!/\w{1}\d{3}/.test(codComune))
throwError(2, badData.printf('Codice Comune'));
if (!/(M|F){1}/.test(sesso)) throwError(2,
    badData.printf('Sesso'));
if (!isDate(dataNasc)) throwError(2,
    badData.printf('Data di nascita'));
return new CodiceFiscale
(nome,cognome,sesso,codComune,dataNasc);
}
```

A questo punto iniziamo con le funzioni per l'elaborazione del codice fiscale.
Iniziamo dalla parte che riguarda il cognome:

```
CodiceFiscale.prototype.getCognomePart =
                                    function (){
    var consonants =
        this.cognome.extractLetters('c','u');
    var vocals =
        this.cognome.extractLetters('v','u');
    return (
consonants.concat(vocals,['X','X','X'])
    ).slice(0,3).join('');
}
```

In pratica si crea un array che contiene di seguito prima le consonanti, poi le vocali e infine 3 caratteri "X" e si ritornano i primi tre caratteri uniti in una stringa.
Analogamente si procede con il nome tenendo presente che qui dovremo ordinare le consonanti mettendo la terza e la quarta prima della seconda:

```
CodiceFiscale.prototype.getNomePart = function
                                        (){
    var consonants =
        this.nome.extractLetters('c','u');
    var vocals =
        this.nome.extractLetters('v','u');
    var ordered = [];
    var rest = [];
    for(var i=0;i<consonants.length;i++){
        if(i==0||i==2||i==3)
            ordered.push(consonants[i]);
        else rest.push(consonants[i]);
    }

    return (
    ordered.concat(rest,vocals,['X','X','X'])
    ).slice(0,3).join('');
}
```

Per ragioni di spazio riportiamo solo la dichiarazione delle altre funzioni che sono:

```
CodiceFiscale.prototype.getDatePart = function
                                        (){
/*
Recupera la parte Giorno Mese e Anno del codice
fiscale

*/
}
CodiceFiscale.prototype.getControlChar = function
                                        (s){
/*
Calcola il carattere di controllo finale
*/
}
```

Il risultato sarà dato dal metodo toString() che non farà altro che assemblare i risultati degli altri metodi:

```
CodiceFiscale.prototype.toString = function (){
    var result = '';
    result += this.getCognomePart();
    result += this.getNomePart();
    result += this.getDatePart();
    result += this.codComune;
    result += this.getControlChar(result);
    return result;
}
```

Nella pagina che ospita l'applicazione utilizzeremo l'oggetto CodiceFiscale passandogli i dati che derivano dall'interfaccia utente:

Fig. 8: Risultato finale

```
function eseguiCalcolo (){
/* N.B. $E è un helper definito altrove */
        var cf = null ;
        cf=
            CodiceFiscale.createInstance (
                $E.value('nome'),
                $E.value('cognome'),
                $E.value('sex'),
codComuneAutocomplete.value.code,
                dataNasc.value)
                );
        addResult(cf);
}
```

Il metodo eseguiCalcolo passa poi l'oggetto creato alla funzione addResult che, in questo caso, utilizza l'oggetto per creare la riga di una tabella:

```
function addResult (cf){
        $E.show("result");
        var oTable = $("resultTable");
        var td,  tr = oTable.insertRow(-
                                    1);
        td = tr.insertCell(-1);
                td.innerHTML= cf.nome;
        td = tr.insertCell(-1);
                td.innerHTML= cf.cognome;
        td = tr.insertCell(-1);
                td.innerHTML= cf.sesso;
        ...
}
```

In figura 8 possiamo vedere il risultato finale della nostra applicazione con la tabella alla quale vengono appese le nuove righe con i codici fiscali calcolati.

CONCLUSIONI

La tecnica non è certo paroganabile ad Ajax, che nasconde in se una complessità decisamente più elevata ed un numero di opportunità sicuramente maggiore, tuttavia sicuramente rappresenta un ottimo workaround quando si devono caricare elevate quantità di dati in una pagina HTML. Certo è necessario comprendere a fondo la logica con cui i vari frame sono innestati, tuttavia questo può rappresentare solo un ostacolo iniziale, una volta compreso lo schema di funzionamento delle varie pagine, può essere utilizzato come modello per un numero veramente elevato di casi.

Francesco Smelzo

GRANDE FRATELLO CON IL GPS

ECCO COME USARE IL SUPPORTO AL GPS DI WINDOWS MOBILE 5 ED INVIARE LA NOSTRA POSIZIONE AD UN SERVER WEB CHE, TRAMITE L'USO DEI SERVIZI MESSI A DISPOSIZIONE DA GOOGLE MAPS O MICROSOFT VIRTUAL EARTH, LA TRACCIA SU UNA MAPPA

Lo scopo del presente articolo è quello di realizzare una soluzione di tracciamento che possiamo schematicamente osservare nella **Figura 1**.

Nella prima parte dell'articolo abbiamo descritto l'applicazione mobile installata su di un dispositivo che monta il sistema operativo Windows Mobile 5 o Windows Mobile 6 e dotato di una antenna per la ricezione delle informazioni satellitari. Abbiamo altresì analizzato la struttura del web service a cui vengono inviate le tracce rilevate dall'apparato GPS, e la struttura di una classe wrapper che incapsula la logica di persistenza delle posizioni su di una base dati.

In questa seconda parte, non ci resta che progettare e realizzare l'applicazione da installare nella nostra ipotetica Centrale di Controllo.

LA CENTRALE DI CONTROLLO

Tra le varie possibilità di implementazione, scegliamo una soluzione che soddisfa ai seguenti requisiti funzionali:

- Innanzitutto vogliamo che la soluzione sia basata sul web;
- In primo piano ci deve essere una mappa in cui vengono rappresentate le posizioni dei diversi dispositivi;
- Le posizioni devono "fluire" sulla mappa, ovvero dobbiamo riprodurre sulla mappa i movimenti dei dispositivi, in tempo reale, quindi è auspicabile produrre una soluzione basata su AJAX (Asynchronous JavaScript and XML).

Come primo problema dobbiamo trovare il modo di rappresentare su di una mappa le posizioni dei dispositivi. Negli ultimi tempi c'è stata una vera e propria "esplosione" di servizi orientati agli sviluppatori che consentono lo sviluppo di soluzioni di geolocalizzazione; tra le varie possibilità, la nostra soluzione farà uso dei servizi messi a disposizione da due colossi del World Wide Web ovvero Google Maps, messo a disposizione da Google, e Virtual Earth, messo a disposizione da Microsoft.

Il lettore che volesse approfondire anche i servizi messi a disposizione da altri fornitori, può trovare, nel box laterale, le indicazioni per raggiungere le pagine dedicate agli sviluppatori da ViaMichelin e Yahoo.

Le condizioni di uso dei vari servizi variano da caso a caso ma sono sempre chiare e dettagliate;

Fig. 1: Schema della soluzione

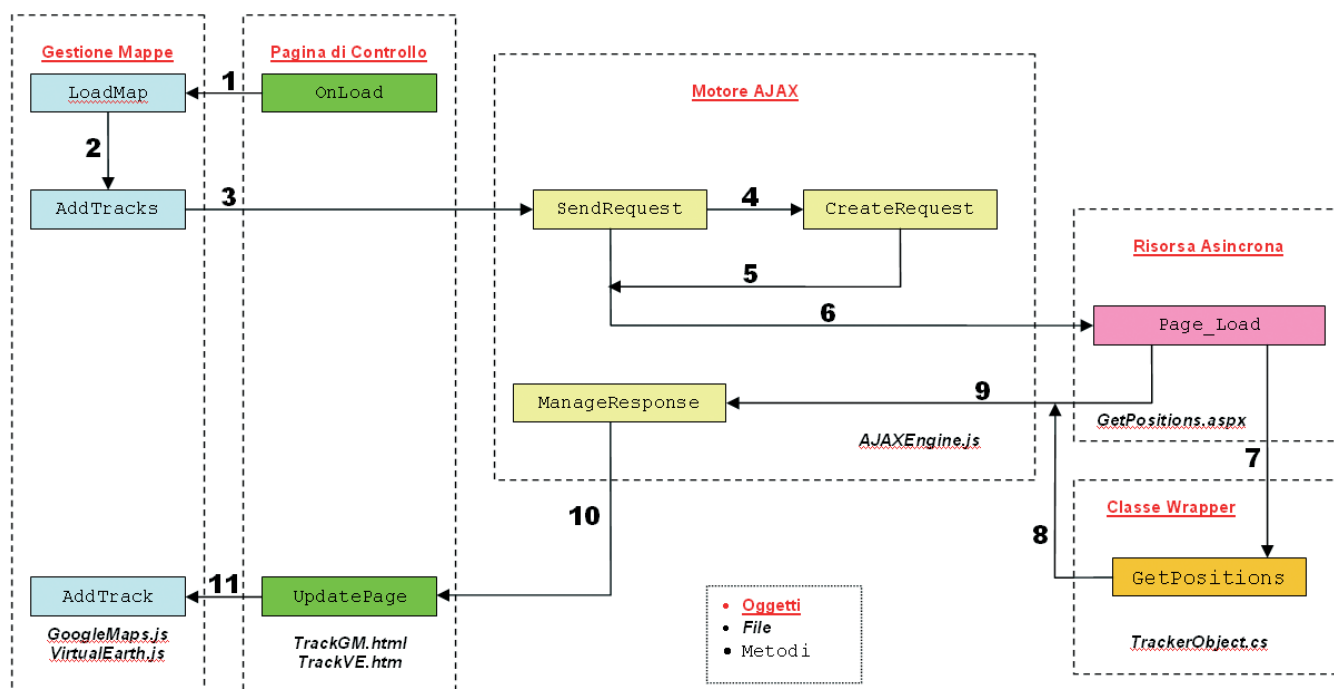



Fig. 2: Flusso delle operazioni nell'applicazione di controllo

normalmente i servizi possono essere usati gratuitamente anche se, in questo caso, le condizioni di uso sono più restrittive. Di solito esiste altresì la possibilità di utilizzare in maniera commerciale il servizio; in questo caso però ogni fornitore ha le sue regole che possono variare notevolmente.
Questa premessa solo per sottolineare che, nel caso in cui volessimo porre in produzione la nostra soluzione, dovremmo leggere attentamente le condizioni di uso del servizio che sceglieremo di utilizzare.
Nell'esempio che stiamo analizzando, l'applicazione della nostra Centrale di Controllo sviluppata con Google Maps è contenuta nel file *TrackGM.html*, quella sviluppata con Virtual Earth è contenuta nel file *TrackVE.htm*.

GLI ELEMENTI DEL SISTEMA

Anche se l'applicazione di controllo è rappresentata da un'unica pagina, la struttura "a strati" della soluzione e l'adozione di servizi esterni e tecnologie quali AJAX, fanno sì che l'analisi di tutti i componenti debba essere necessariamente preceduta da uno schema, che riassume sia gli elementi in gioco sia il flusso delle operazioni che avvengono quando accediamo alla pagina di controllo.
Nella **Figura 2** abbiamo rappresentato la sequenza di operazioni che vengono scatenate all'atto dell'apertura della pagina di controllo; i vari blocchi sono i metodi che vengono invocati in sequenza; abbiamo altresì raggruppato i diversi metodi in modo da poter individuare sia il componente logico che viene invocato, sia il relativo file in cui questi metodi sono memorizzati.
Per meglio comprendere i vari elementi che verranno descritti di seguito, è importante descrivere la sequenza delle operazioni:

1) La pagina di controllo viene caricata e viene scatenato l'evento OnLoad dell'elemento BODY che invoca il metodo LoadMap del componente di gestione della mappe il quale configura i parametri iniziali della mappa.
2) Il metodo LoadMap crea un timer che ogni 3 secondi invoca il metodo AddTracks che interroga, in modo asincrono, una risorsa remota che fornisce i dati dei diversi dispositivi.
3) Il metodo AddTracks invoca il metodo SendRequest che rappresenta l'interfaccia "in ingresso" del motore AJAX che gestisce le richieste asincrone.
4) Il metodo SendRequest invoca il metodo CreateRequest per la creazione di un oggetto di tipo XMLHttpRequest utilizzato per gestire le chiamate asincrone.
5) Il metodo CreateRequest ritorna il giusto oggetto che dipende dal tipo di navigatore che stiamo usando dato che diversi navigatori supportano in modo diverso l'oggetto

XMLHttpRequest.

6) Il metodo SendRequest invoca la risorsa remota attraverso il metodo open dell'oggetto di tipo XMLHttpRequest.
7) Nel nostro esempio, la risorsa remota è la pagina GetPositions.aspx di cui viene eseguito il metodo Page_Load che crea una istanza della classe wrapper TrackerObject ed invoca il suo metodo GetPositions.
8) Il metodo GetPositions ritorna una stringa che contiene i dati relativi ai dispositivi che vogliamo tracciare.
9) La risorsa remota torna la risposta al chiamante invocando il metodo ManageResponse del motore AJAX.
10) Il metodo ManageResponse passa, a sua volta, il risultato proveniente dalla risorsa remota, al metodo UpdatePage della pagina di controllo.
11) Il metodo UpdatePage invoca per ogni dispositivo, il metodo AddTrack del componente di gestione delle mappe che, infine, disegna sulla mappa un indicatore che rappresenta la posizione del dispositivo.

Per implementare il precedente meccanismo, a livello di codice, dobbiamo innanzitutto inserire nella pagina di controllo i riferimenti ad alcuni componenti utilizzati nello schema, ovvero il riferimento al servizio che intendiamo utilizzare, il riferimento al motore AJAX ed il riferimento al componente di gestione delle mappe. In testa alla pagina andremo allora a porre alcune dichiarazioni che fanno riferimento al relativo codice Javascript; per Microsoft Virtual Earth la dichiarazione è:

```
<!-- Riferimento al servizio Microsoft Virtual Earth -->
<script
src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=5"></script>
<!-- Riferimento al Motore AJAX -->
<script src="AJAXEngine.js"></script>
<!-- Riferimento al Gestore della mappa -->
<script src="VirtualEarth.js"></script>
```

Per Google Maps invece:

```
<!-- Riferimento al servizio Google Maps -->
<script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=chiave"
type="text/javascript"></script>
<!-- Riferimento al Motore AJAX -->
<script src="AJAXEngine.js"></script>
<!-- Riferimento al Gestore della mappa -->
<script src="GoogleMaps.js"></script>
```

Dobbiamo notare che per l'utilizzo dei servizi forniti da Google Maps è necessario richiedere una chiave di attivazione che poi dobbiamo specificare proprio nella dichiarazione del relativo codice Javascript. Dato che questa chiave dipende dall'indirizzo in cui andremo a pubblicare la nostra applicazione, è prevedibile che, così come dichiarato nel file TrackGM.html, il codice non funzioni. Per utilizzare l'esempio fornito è necessario richiedere una propria chiave di attivazione e sostituirla con quella fornita; nel sito dedicato agli sviluppatori (il cui indirizzo è indicato nel box a lato) sono indicate tutte le procedure necessarie per ricevere le propria chiave di attivazione.

Il corpo della pagina è molto semplice dato che tutte le interazioni, avvengono tramite un elemento di tipo div. Per Microsoft Virtual Earth il corpo della pagina è:

```
<body onload="LoadMap();">
 <table width="100%">
 <tr>
  <td colspan="2" align="left"><h1>Traccia la tua
     Flotta con Virtual Earth</h1></td>
  <td align="center" valign="middle" rowspan="2">
     <div id='myMap' style="position:relative;
     width:600px; height:600px;" ></div>
  </td>
 </tr>
 <tr>
  <td align="center" valign="middle"><img
     src="tracker.gif" /></td>
 </tr>
 </table>
</body>
Per Google Maps invece:
<body onload="LoadMap()">
 <table width="100%">
 <tr>
  <td colspan="2" align="left"><h1>Traccia la tua
     Flotta con Google Maps</h1></td>
  <td align="center" valign="middle" rowspan="2">
     <div id='map' style="position:relative;
     width:600px; height:600px;" ></div>
  </td>
 </tr>
 <tr>
  <td align="center" valign="middle"><img
     src="tracker.gif" /></td>
 </tr>
 </table>
</body>
```

Da notare la definizione del gestore LoadMap che, in occasione dell'evento di apertura della

pagina di controllo, scatena tutto il processo. Dato che abbiamo come riferimento lo schema dalla Figura 2, possiamo passare all'analisi dei singoli componenti che vengono interessati.

IL MOTORE AJAX

Il motore AJAX che utilizziamo nelle nostre pagine è contenuto nel file *AJAXEngine.js*. Chiaramente la descrizione dei meccanismi che regolano le chiamate asincrone effettuate tramite AJAX, esula dagli scopi del presente articolo. In questo contesto è interessante descrivere semplicemente come abbiamo implementato l'invocazione della risorsa remota e la relativa gestione della risposta. Alla base di tutto c'è l'oggetto *XMLHttpRequest* che ormai tutti i navigatori di ultima generazione supportano. Il nostro motore è composto da tre funzioni che rispettivamente creano l'oggetto *XMLHttpRequest*, inviano la richiesta asincrona e gestiscono la risposta.

La funzione *CreateRequest* si occupa di creare un oggetto di tipo *XMLHttpRequest*. I diversi navigatori disponibili supportano l'oggetto *XMLHttpRequest* in modo differente. Alcuni hanno il supporto nativo, come, ad esempio, Mozilla Firefox. I navigatori di casa Microsoft invece supportano l'oggetto tramite la tecnologia ActiveX con due diverse versioni dell'oggetto dipendenti dalla versione del navigatore.

Dato che intendiamo sviluppare una applicazione indipendente dal navigatore, la funzione *CreateRequest* gestisce le diverse situazioni restituendo sempre il giusto oggetto.

```
function CreateRequest()
{
 var request = false;
 try {
   // Supporto Nativo
   request = new XMLHttpRequest();
 } catch (newIE) {
   try {
     // Nuova versione di IE
     request = new
                   ActiveXObject("Msxml2.XMLHTTP");
   } catch (oldIE) {
     try {
       // Vecchia versione di IE
       request = new
                   ActiveXObject("Microsoft.XMLHTTP");
     } catch (error) {
       // Il navigatore NON supporta XMLHttpRequest
       request = false;
     }
   }
 }
 return request;
}
```

L'invio della richiesta è effettuata dalla funzione *SendRequest* che rappresenta l'interfaccia di ingresso del motore e che riceve in input l'indirizzo (compresi gli eventuali parametri) della risorsa da invocare.

Nella prima linea di codice creiamo una istanza dell'oggetto *XMLHttpRequest* invocando la funzione *CreateRequest* appena analizzata. Dobbiamo quindi configurare la richiesta utilizzando il metodo *open* dell'oggetto *XmlHttpRequest* a cui passiamo, come primo parametro, il tipo di richiesta (che può essere "GET" o "POST"); come secondo parametro, l'indirizzo della risorsa web a cui connettersi; come terzo parametro un valore booleano che, se valorizzato a *true*, indica che la richiesta sarà asincrona.

Nel caso in cui la risorsa a cui ci connettiamo necessiti di autenticazione, possiamo utilizzare la versione a 5 parametri del metodo *open*; potremo specificare un nome utente nel quarto parametro e la relativa password nel quinto parametro.

Prima di inviare la richiesta al server dobbiamo ancora definire il meccanismo di "call-back" in modo che il server sia in grado di ritornare le informazioni richieste, al navigatore.

Per ora ci occupiamo di agganciare alla richiesta una funzione, che verrà invocata una volta che i dati saranno disponibili. Dobbiamo valorizzare la proprietà *onreadystatechange* dell'oggetto *XmlHttpRequest* che rappresenta il gestore dell'evento associato alla variazione dello stato della richiesta; valorizziamo la proprietà con il nome della funzione che vogliamo utilizzare come gestore della risposta del server (analizzeremo la funzione a breve).

Come ultima operazione dobbiamo inviare la richiesta al server; utilizziamo il metodo *send* dell'oggetto *XmlHttpRequest* a cui si può passare come opzione un parametro per inviare contenuto al server. È possibile altresì inviare parametri sulla linea di comando, compilando opportunamente l'URL della richiesta.

```
function SendRequest(url)
{
 // Creiamo l'oggetto XmlHttpRequest
 xmlHttpReq = CreateRequest();
 // Configuriamo la richiesta
 xmlHttpReq.open("GET", url, true);
 // Impostiamo la funzione di call-back
 xmlHttpReq.onreadystatechange = ManageResponse
 // Inviamo la richiesta al server
```

GOOGLE MAPS

Per accedere al servizio messo a disposizione da Google puntiamo il nostro navigatore verso l'indirizzo: http://maps.google.com/. L'area dedicata agli sviluppatori è raggiungibile a partire dall'indirizzo: http://www.google.com/apis/maps/.

```
xmlHttpReq.send(null);
}
```

Per gestire la risposta del server alla nostra richiesta, dobbiamo implementare la funzione *ManageResponse* che viene invocata ogni volta che varia lo stato della richiesta.
Lo stato della richiesta può assumere valori che vanno da 0 (*uninitialized*) a 4 (*complete*). Chiaramente l'elaborazione dei dati deve avvenire solo dopo che la richiesta ha raggiunto lo stato "*complete*".
Prima di procedere nell'analisi della risposta, è necessario verificare che l'interazione non abbia generato errori. Attraverso le proprietà status e *statusText* possiamo verificare il codice di ritorno della richiesta HTTP. Il codice 200 (il cui relativo valore testuale è "OK") ci segnala che il colloquio non ha generato errori quindi possiamo elaborare la risposta.
In caso di errore lo stato verrà valorizzato con i valori che a volte riscontriamo durante la navigazione: 500 ("Internal Server Error"), 404 ("Not Found"), ecc.
L'oggetto *XmlHttpRequest* mette a disposizione due proprietà attraverso cui possiamo elaborare la risposta del server: *responseText* contiene la risposta del server in formato stringa mentre *responseXML* contiene la risposta del server in formato XML, in particolare, se la risorsa invocata è un web service, il documento XML ritornato, rappresenta il messaggio SOAP inviato dal server.
Nell'esempio corrente la nostra funzione passerà semplicemente il contenuto della risposta ad una funzione (*UpdatePage*) che rappresenta l'interfaccia di uscita del motore e che dovrà essere implementata al livello della pagina che utilizza il motore AJAX.

```
function ManageResponse()
{
 // Verifichiamo che la richiesta sia completata
 if (xmlHttpReq.readyState == 4)
 {
  // Verifichiamo che il colloquio non abbia generato errori
  if (xmlHttpReq.status == 200)
  {
   // Passiamo la risposta del server ad una funzione
   // che gestirà il risultato al livello della pagina
   UpdatePage(xmlHttpReq.responseText);
  }
 }
}
```

Strutturato in questo modo, il motore AJAX è un componente autocontenuto con una interfaccia di ingresso (SendRequest) ed una di uscita (UpdatePage); ciò ci consente di poterlo utilizzare in ogni occasione in cui vogliamo aggiungere un supporto alle chiamate asincrone nelle nostre applicazioni. Ovviamente non pretendiamo di sostituire i completi framework od oggi disponibili (uno su tutto ASP.NET AJAX), ma sicuramente, con poche linee di codice, siamo in grado di gestire "semplici" interazioni in modo rapido ed "economico".

FACCIAMOLO CON GOOGLE MAPS

Il tracciamento dei dispositivi avviene interagendo con le primitive messe a disposizione dai diversi servizi. Sebbene il risultato sia sostanzialmente identico, i due servizi hanno sintassi diverse quindi è necessario analizzare separatamente il codice Javascript relativo.
Per interagire con Google Maps, nel file *TrackGM.html* poniamo un riferimento al file *GoogleMaps.js* che rappresenta il componente di gestione della mappa rappresentato in **Figura 2**.
Il file contiene tre funzioni; la prima, *LoadMap*, viene invocata in fase di caricamento della pagina di controllo e non fa altro che creare un nuovo oggetto (*map*) che useremo per interagire con la mappa. Dopo aver aggiunto alcuni controlli e centrato la mappa (la posizione dell'esempio centra la mappa su Ancona), attiviamo un timer per effettuare l'aggiornamento delle posizioni.
Il metodo Javascript *setInterval*, viene utilizzato per invocare una funzione ogni volta che trascorre un intervallo di tempo espresso in millisecondi. In questo caso invochiamo, ogni 3 secondi, la funzione *AddTracks* che non fa altro che invocare una risorsa remota attraverso la funzione *SendRequest* del motore AJAX analizzato in precedenza.

```
function LoadMap()
{
 // Creiamo la mappa
 map = new
         GMap2(document.getElementById("map"));
 // Aggiungiamo alcuni controlli
 map.addControl(new GSmallMapControl());
 map.addControl(new GMapTypeControl());
 // Centriamo la mappa
 map.setCenter(new GLatLng(43.6166483,
                                       13.5188783), 17);
 // Attiviamo il timer per l'aggiornamento
 timerId = setInterval("AddTracks()", 3000);
}
function AddTracks()
{
```

NOTA

ALTRI SERVIZI DI GEOLOCALIZZAZIONE

La pagina dedicata agli sviluppatori del servizio ViaMichelin è raggiungibile all'indirizzo: http://dev.viamichelin.com/.
La pagina dedicata agli sviluppatori del servizio Yahoo Maps è raggiungibile all'indirizzo: http://developer.yahoo.com/maps/.

```
// Invoca la funzione del motore AJAX
SendRequest('GetPositions.aspx');
}
```

L'ultima funzione del file *GoogleMaps.js* si occupa di porre sulla mappa un indicatore che rappresenta un dispositivo da tracciare. La funzione *AddTrack*, viene invocata dalla funzione *UpdatePage* che rappresenta l'interfaccia di uscita del motore AJAX. La funzione riceve in ingresso quattro parametri: l'identificativo del dispositivo, la sua descrizione e le sue coordinate geografiche rappresentate nella notazione decimale. Utilizzando le primitive del servizio, creiamo innanzitutto un oggetto che rappresenta il "punto geografico" corrispondente alle coordinate passate in ingresso. Quindi creiamo un oggetto "indicatore" (rappresentato da una icona) che verrà posizionato sulla mappa proprio nel punto creato alla linea precedente. Per aumentare l'esperienza del fruitore dell'applicazione di controllo, associamo all'indicatore una funzione che gestisce l'evento del clic in modo che venga aperta una finestra informativa che, nella fattispecie, mostrerà l'identificativo e la descrizione del dispositivo rappresentato dall'indicatore.
Come ultima operazione, aggiungiamo l'indicatore alla mappa.

```
function AddTrack(idDevice, deviceDescription,
                                        latitude, longitude)
{
 // Creiamo un punto goegrafico
 var point = new GLatLng(latitude, longitude);
 // Associamo al punto un indicatore predefinito
 var marker = new GMarker(point);
 // Aggiungiamo all'indicatore l'evento "click"
 // per mostrare un messaggio informativo
 GEvent.addListener(marker, "click",
     function() {
      marker.openInfoWindowHtml("Dispositivo: " +
       idDevice + " <br/><b>" +
       deviceDescription + "</b>");  });
 // Aggiungiamo l'indicatore alla mappa
 map.addOverlay(marker);
}
```

FACCIAMOLO CON VIRTUAL EARTH

La pagina di controllo basata su Microsoft Virtual Earth è contenuta nel file *TrackVE.htm* mentre il relativo codice del componente di gestione della mappa del servizio è contenuto nel file *VirtualEarth.js*.
Anche in questo caso, l'interazione con la mappa avviene attraverso tre funzioni. La prima (*LoadMap*) si occupa di creare l'oggetto con cui andremo ad interagire (comprese le impostazioni iniziali ed il punto in cui viene centrata la mappa) e scatena il temporizzatore per l'aggiornamento delle tracce. La seconda funzione (*AddTracks*) non fa altro che invocare il motore AJAX con le identiche modalità dell'esempio visto in precedenza. L'ultima funzione (*AddTrack*) si occupa dell'effettiva creazione dell'indicatore e del relativo posizionamento

```
function LoadMap()
{
 // Creiamo la mappa
 map = new VEMap('myMap');
 // Centriamo la mappa
 map.LoadMap(new VELatLong(43.6166, 13.5189),
                                  17 ,'h' ,false);
 // Definiamo il tipo di mappa da visualizzare
 map.SetMapStyle(VEMapStyle.Road);
 // Nascondiamo alcuni controlli predefiniti
 map.HideDashboard();
 // Attiviamo il timer per l'aggiornamento
 timerId = setInterval("AddTracks()", 3000);
}
function AddTracks()
{
 // Invoca la funzione del motore AJAX
 SendRequest('GetPositions.aspx');
}
function AddTrack(idDevice, deviceDescription,
                                        latitude, longitude)
{
 // Creiamo un punto goegrafico
 var PPposition = new
                        VELatLong(parseFloat(latitude),
                                   parseFloat(longitude));
 // Associamo al punto un indicatore predefinito
 var shape = new VEShape(VEShapeType.Pushpin,
                                              PPposition);
 // Associamo all'indicatore le informazioni sul
                                               dispositivo
 shape.SetTitle('Dispositivo ' + idDevice);
 shape.SetDescription(deviceDescription);
 // Aggiungiamo l'indicatore alla mappa
 map.AddShape(shape);
}
```

A causa della sostanziale corrispondenza delle funzioni svolte dal componente possiamo limitarci ad appuntare le differenze sintattiche tra i due servizi.

LA RISORSA REMOTA

Nel nostro esempio, la risorsa remota è rappre-

NOTA

MICROSOFT VIRTUAL EARTH

Per accedere alle informazioni relative al servizio messo a disposizione da Microsoft puntiamo il nostro navigatore verso l'indirizzo: http://www.microsoft.com/virtualearth/default.mspx da cui è accessibile il servizio vero e proprio di ricerca su mappa che è pubblicato all'indirizzo http://local.live.com/. L'area dedicata agli sviluppatori è raggiungibile a partire dall'indirizzo: http://dev.live.com/virtualearth/

sentata dalla pagina GetPositions.aspx che ritorna una stringa opportunamente formattata che contiene l'elenco delle informazioni afferenti ai dispositivi da tracciare.
Se andiamo ad analizzare il codice "dietro" la pagina, notiamo che l'operazione svolta è banale a causa della stratificazione del codice che abbiamo imposto nello sviluppo della soluzione. All'atto del caricamento della pagina viene creata una istanza della classe TrackerObject che abbiamo analizzato all'inizio dell'articolo. Quindi viene invocato il suo metodo GetPositions che restituisce la stringa di dati che non dobbiamo far altro che passare al chiamante tramite il metodo Write dell'oggetto Response. Prima di inviare la risposta al chiamante è opportuno impostare alcuni parametri al fine di evitare problemi con le cache dei navigatori.

```
protected void Page_Load(object sender
                                        EventArgs e)
{
 // Creiamo una istanza della classe astratta
 TrackerObject.TrackerObject oTracker = new
                          TrackerObject.TrackerObject();
 // Ricaviamo le posizioni dei dispositivi
 string sPositions = oTracker.GetPositions();
 // Configuriamo la cache
 Response.CacheControl = "no-cache";
 Response.AddHeader("Pragma", "no-cache");
 Response.Expires = -1;
 // Inviamo la risposta al chiamante
 Response.Write(sPositions);
 Response.End();
}
```

La pagina *GetPositions.aspx* termina il suo compito subito dopo aver inviato la risposta al chiamante; come ultima riga di codice, invochiamo il metodo *End* dell'oggetto *Response* che invia al chiamante tutto il contenuto (eventuale) del buffer e termina l'esecuzione della pagina.

LA FUNZIONE UPDATEPAGE

La funzione *UpdatePage*, implementata a livello delle pagine di controllo, riceve in input il risultato dell'invocazione della risorsa asincrona; se facciamo un passo indietro e torniamo alla prima parte del presente articolo, ci ricordiamo che il risultato atteso è una stringa tipo:

```
1^Primo
        Dispositivo^43,61783^13,52374|2^Secondo
                Dispositivo^44,43323^13,44238| . . .
```

Dopo aver eliminato tutte le tracce presenti sulla mappa, creiamo un array in cui ogni elemento rappresenta le informazioni relative ad un dispositivo; quindi scorriamo l'array e, dopo aver isolato ulteriormente le informazioni relative al singolo dispositivo invochiamo la funzione *AddTrack* per posizionare la relativa traccia sulla mappa.
Il servizio fornito da Google offre una interessante primitiva (*getBounds*) che ci consente di ricavare i contorni visibili della mappa e verificare se un punto è contenuto nei suddetti contorni; di conseguenza riusciamo facilmente a capire se la traccia di un dispositivo esce dalla zona della mappa attualmente visibile e, di conseguenza, riusciamo a centrare nuovamente la mappa sul dispositivo in modo da "seguirlo" letteralmente. Nell'esempio oggetto dell'articolo si è scelto, a puro titolo di esempio, di "seguire" il primo dispositivo della serie.

```
function UpdatePage(result)
{

// Eliminiamo tutte le tracce

 map.clearOverlays();

 // Creiamo un array in cui ogni elemento
 // rappresenta un dispositivo

 arrDevices = result.split("|");
 for(var i=0; i<arrDevices.length-1; i++)
 {

  // Creaimo un array in cui suddividiamo
  //le informazioni relative ad un singolo dispositivo

  arrPosition = arrDevices[i].split("^");

  // Se stiamo gestendo il primo dispositivo

  if(i == 0)
  {

   // Ricaviamo i bordi della mappa
   var bounds = map.getBounds();

// Creiamo il punto in cui è l
// localizzato il primo dispositivo

   var positionDevice1 = new GLatLng(arrPosition[2],
                                       arrPosition[3])
   // Se il primo dispositivo è fuori dalla mappa ...
   if(!bounds.contains(positionDevice1))
   {
    // Centriamo la mappa sul dispositivo
    map.setCenter(positionDevice1, 17);
```

```
      }
    }
    // Aggiungiamo alla mappa l'indicatore del
                                          dispositivo
    AddTrack(arrPosition[0], arrPosition[1],
                          arrPosition[2], arrPosition[3]);
  }
}
```

TESTIAMO LA SOLUZIONE

Siamo ormai giunti al termine, non dobbiamo far altro che distribuire i componenti della soluzione e provare il tutto.
Chiaramente la complessità della soluzione non consente una verifica rapida. Se non altro perché abbiamo bisogno di almeno due agenti: il controllore accede alla pagina di controllo e visualizza i movimenti del volontario che, armato di dispositivo mobile con antenna di ricezione satellitare e programma *TackerMobile* attivo e configurato, gira per la nostra città e si lascia tracciare.
Ovviamente tutta la parte “web” della soluzione (pagina di controllo e web service) devono essere installate in un server “raggiungibile” dal dispositivo mobile ed in grado a sua volta di accedere alla rete in modo da raggiungere i servizi di Google e Microsoft.
Se supponiamo comunque di aver installato, configurato ed avviato correttamente tutti i componenti, se ci poniamo davanti alla pagina di controllo, vediamo qualcosa di simile alla pagina mostrata in Figura 3.

L’icona di colore rosa rappresenta il dispositivo che stiamo tracciando. Con un gradevole effetto dinamico vediamo l’icona spostarsi sulla mappa in modo fluido (a causa di AJAX). Nella versione che utilizza il servizio Google Maps, se facciamo clic sull’icona, un fumetto ci fornisce alcune informazioni sul dispositivo.

CONCLUSIONI

Lo sviluppo tecnologico di dispositivi e sistemi software ormai ci consente di realizzare applicazioni molto complesse con una relativa facilità.
Nell’articolo appena letto abbiamo avuto modo di completare lo sviluppo di una soluzione completa per il tracciamento di dispositivi in movimento sul territorio.
Abbiamo ipotizzato una dotazione di dispositivi mobili basati sul sistema operativo Windows Mobile 5 o Windows Mobile 6, dotati di apparati per la ricezione satellitare (antenne GPS) e dotati di supporto telefonico per la comunicazione della posizione alla Centrale di Controllo.
Nella prima parte dell’articolo abbiamo sviluppato sia l’applicazione mobile sia il web service a cui, la suddetta applicazione, invia i dati relativi al posizionamento. Al fine di strutturare la soluzione in più “strati logici”, abbiamo altresì sviluppato una classe wrapper che incapsula la logica di persistenza delle posizioni su di una base dati.
In questa seconda parte abbiamo sviluppato l’applicazione di controllo che è basata su servizi di geolocalizzazione forniti da Google o Microsoft e sfrutta la potenzialità di AJAX per gestire il movimento delle tracce.
Se vogliamo ipotizzare qualche utilizzo della nostra soluzione, possiamo pensare, ad esempio, alla sala operativa della Polizia Municipale della nostra città dove, in un maxischermo, vengono visualizzate le posizioni delle pattuglie distribuite sul territorio in modo da poter dirigere l'unità più vicina in caso di sinistro.
Un altro utilizzo "civico" lo possiamo ipotizzare per una azienda di trasporto pubblico locale dove, a fronte della posizione corrente degli autobus, si possono dare informazioni ai passeggeri in attesa alle fermate, sui tempi di attesa del proprio mezzo.
Anche le aziende di trasporto merci, piuttosto che i pony-express possono trarre giovamento dal sapere in tempo reale la posizione dei diversi corrieri per soddisfare in modo più rapido le continue richieste dei clienti.
Da questi pochi esempi possiamo già capire che, ancora una volta, il limite all'utilizzo di queste tecnologie avanzate è la nostra fantasia.

Oscar Peli

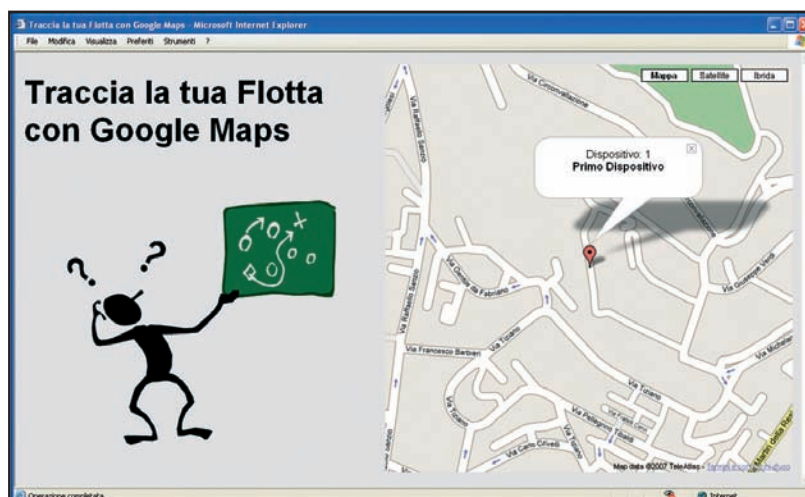



***Fig. 3:** La pagina di controllo (versione Google Maps)*

TRACCE ISO DAI NOSTRI PROGRAMMI

PERCHÈ UTILIZZARE UN SOFTWARE ESTERNO PER MASTERIZZARE I DATI? CREIAMO UN'APPLICAZIONE IN GRADO DI GESTIRE LE TRACCE ISO. USIAMOLE COME SE FOSSERO UNA PARTE DEL NOSTRO FILE SYSTEM, IN MODO DEL TUTTO TRASPARENTE

Sempre più spesso si rende necessario dover lavorare con delle immagini di CD, DVD, Filesystem o altro. L'utilità di avere queste immagini che "rispecchiano" il contenuto di tali dispositivi di memoria di massa è quella di riuscire ad effettuare dei backup completi e salvarli su un file, astraendo quindi il contenitore dei dati dal vero e proprio mezzo fisico, per poi ridistribuirlo e ripristinarlo nelle maniere che più si ritengono opportune.
Esiste una serie molto vasta di programmi che permettono di generare un'immagine a seconda del target e viceversa, cioè generare un CD o un DVD a partire da un file immagine (operazione che viene comunemente detta di masterizzazione). Nero è sicuramente il più famoso e conosciuto tra tutti questi software ma non è il solo.

siamo testare solamente dopo che l'immagine è stata masterizzata. Sarebbe invece di grande utilità poter "ingannare" il nostro computer in modo tale da masterizzare l'immagine in questione non su un DVD o CD, bensì su una porzione del nostro hard-disk che venga però riconosciuta dal sistema operativo come un'unità fisica..

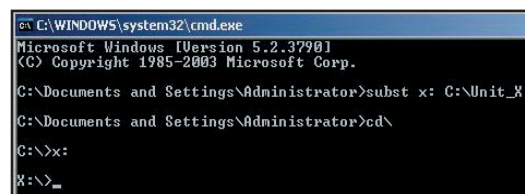



Figura 1: montaggio di un'unità virtuale con etichetta X.

UN ESEMPIO PRATICO

Uno degli innumerevoli scenari possibili potrebbe essere questo:
supponiamo di dilettarci per hobby di video processing e che solitamente i nostri amici ci chiedano di abbellire i lori filmati con sottotitoli, effetti etc. I nostri amici potrebbero fornirci l'immagine del loro DVD (magari mettendolo su un nostro server ftp) in modo da poterlo masterizzare su un nostro DVD vergine. Al termine delle modifiche creeremo una nuova immagine che conterrà il filmato ritoccato da restituire al nostro amico.
Tale situazione è nella pratica molto inefficiente e costosa, infatti ogni volta siamo costretti a masterizzare un DVD per poi buttarlo appena terminato il lavoro. Così facendo sprechiamo sia del tempo, poiché dobbiamo aspettare di portare a termine la masterizzazione di un DVD, sia un DVD vergine per ogni lavoro, poiché una volta terminato il lavoro, del DVD masterizzato non sappiamo più che farcene.
Uno scenario molto simile a quello precedentemente descritto lo abbiamo incontrato un po' tutti quando scarichiamo un'immagine da un programma P2P. Infatti molte volte non possiamo avere la certezza del contenuto di ciò che abbiamo scaricato e lo pos-

PIANIFICAZIONE DELL'APPLICAZIONE

Cominciamo con il fissare una serie di punti guida che ci serviranno per sviluppare la nostra applicazione. Per poter realizzare quanto sopra detto avremo bisogno di:

1 – montare e smontare delle unità senza in alcun modo andare a modificare le tabelle di partizione del nostro hard disk.

2 – leggere un'immagine (ISO, BIN-CUE etc...) ed estrarre il suo contenuto sull'unità virtuale.

3 – navigare l'unità virtuale creata come qualsiasi altra unità "reale".

CREARE TANTE UNITÀ SENZA CREARE NESSUNA PARTIZIONE

Il primo ostacolo da superare è quella di creare o ricavare un'unità aggiuntiva che fungerà da "DVD virtuale". Una prima idea potrebbe essere quella di partizionare nuovamente il nostro hard disk in modo ta-

Conoscenze richieste
Conoscenza degli ambienti di programmazione Microsoft .net

Software
Microsoft Robotics Studio 1, .NET Framework 3, Visual Studio 2005 o Kit Visual Express C# o VB.net ad libitum

Impegno

Tempo di realizzazione

le da ritagliare un po' di spazio (pari almeno a quello di un DVD) per la nostra unità aggiuntiva e dedicare quest'ultima solamente allo scopo di unità "fantoccio" che emuli un DVD. Questo tipo di approccio presenta almeno due aspetti macroscopicamente negativi che ci spingono alla ricerca di una possibile soluzione più efficiente:

1 – le operazioni di partizionamento sono sempre considerate altamente rischiose; basterebbe questo a farci desistere dal nostro intento.

2 – sarebbe uno spreco riservare spazio ad un'unità che in realtà utilizzeremo una volta ogni tanto e di cui una volta finite tali operazioni non sapremmo più che farcene.

Fortunatamente esiste un comando dos che si chiama subst che ci permette proprio di mappare un'unità virtuale su una directory. Per un momento quindi svestiamo i panni di programmatori e diamo un'occhiata ai vecchi comandi dos invocabili dal prompt. La sintassi del comando subst è la seguente:

1 – per montare una nuova unità:
SUBST [unità:] [percorso]

2 – per smontare un'unità virtuale precedentemente creata:
SUBST /D [unita':]

Per comprendere meglio di cosa si tratta facciamo qualche esempio. Naturalmente trattandosi di un comando DOS avrete bisogno di lanciare la shell, oppure digitare direttamente il comando cliccando su Start, poi su Esegui.
Dobbiamo innanzitutto scegliere una cartella che verrà considerata come cartella "radice" dell'unità virtuale, ad esempio sotto C: io ho creato una directory chiamata Unit_X. A questo punto se volessimo creare un'unità con label X: basterà digitare:

```
subst X: C:\Unit_X
```

Naturalmente è necessario che l'unità X non esista già, altrimenti il comando fallirà. Se ora andate su "Risorse del computer" e, se tutto è andato a buon fine, avrete una nuova unità X.
Questa "partizione fittizia" avrà le dimensioni dell'hard disk della partizione contenente la cartella originaria (nel nostro caso C:\Unit_X) . Ovviamente se occupate spazio su c:\ , lo occuperete anche su x: e viceversa (nonostante non aggiungiate nulla nella cartella C:\Unit_X) visto che comunque al livello pratico i cluster occupati sono sempre gli stessi; di conseguenza se si formatta la partizione originaria scompare anche l'unità virtuale. In figura 1 trovate un esempio di quanto detto.

Nulla è stato ancora modificato sulla tabella delle partizioni, per cui se riavviate il sistema operativo non troverete più l'unità fittizia. Per rendere permanente una configurazione così creata è possibile scrivere un file batch contenente le istruzioni desiderate e linkarlo sotto "esecuzione automatica", ma ciò esula dagli scopi di quest'articolo.
Analogamente se volessimo rimuovere tale unità (senza dover riavviare il sistema operativo) basterà eseguire il comando:

```
subst /D X:
```

E' superfluo ribadire che, anche in questo caso, la scomparsa dell'unità non implica la scomparsa dei dati che invece rimarranno intatti sotto la directory

```
C:\Unit_X
```

ESEGUIRE I PROGRAMMI COME THREAD .NET

Ora che siamo venuti a conoscenza dell'esistenza di un comando DOS che fa al caso nostro sarà necessario integrarlo all'interno della nostra applicazione.
Quest'ultima verrà implementata in C#, per cui, più in generale, sarà necessario lanciare un processo nativo (cioè del sistema operativo) all'interno di un processo .NET. A questo proposito esiste già una classe .NET che permette di avere un buon controllo sui processi esterni. Con "buon controllo" intendo che, tra le altre cose, si possono eseguire le seguenti operazioni:

- lanciare un processo esterno (comando, applicazione etc...) come se fosse un thread parallelo
- "attaccarsi" al processo in modo tale da bloccare l'esecuzione del codice seguente finché il processo stesso non termina
- ridirigere lo standard output, input ed error all'interno dell'applicazione stessa, così che sia possibile "catturare" tutti i messaggi che il processo esterno invia (sull'output e/o sull'error) o inviarne noi sull'input.
- Verificare l'exit code del processo stesso in modo tale da poterne verificare se quest'ultimo ha terminato il proprio compito con successo oppure no.

La classe in questione si chiama propria *Process* e si trova all'interno del namespace *System.Diagnostics*. Nel corso di quest'articolo parleremo diffusamente di questa classe. Cominciamo però con il dare un'occhiata alla **figura 2** che illustra come si presenterà la GUI dell'applicazione che stiamo sviluppando.
Come potete notare sulla sinistra è stato posto un controllo grafico (vedi nota) che ricorda l'approccio

Figura 2: Interfaccia grafica dell'applicazione Image Viewer.

user-friendly del resource explorer di windows, a destra invece sono stati piazzati tre bottoni:

1. monta unità virtuale
2. smonta unità virtuale
3. estrai immagine

Veniamo quindi al codice sottostante il primo bottone:

```
private void buttonMount_Click(object sender,
                                        EventArgs e)
  {
    if (textBoxLabel.Text == string.Empty)
    {
      MessageBox.Show(this, "you have to
         enter a label for the virtual unit", "error",
  MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }
    if (folderBrowserDialog1.ShowDialog(this) !=
                                   DialogResult.OK)
      return;

    Process subst = new Process();
    subst.StartInfo.FileName = "subst";
    subst.StartInfo.Arguments =
                        textBoxLabel.Text + ": " +
               folderBrowserDialog1.SelectedPath;
    subst.Start();
    subst.WaitForExit();

    if (subst.ExitCode != 0)
      MessageBox.Show(this, "I cannot create
        this unit", "error!", MessageBoxButtons.OK,
                            MessageBoxIcon.Error);
    fileSystemListView1.RefreshView();
  }
```

La prima cosa che facciamo è controllare se l'utente abbia effettivamente specificato una lettera per identificare l'unità virtuale che vuole creare, se così non fosse un messaggio d'errore verrà visualizzato ed il codice successivo non sarà eseguito.

```
    if (textBoxLabel.Text == string.Empty)
    {
      MessageBox.Show(this, "you have to
         enter a label for the virtual unit", "error",
  MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }
```

Naturalmente il controllo TextBox ha impostata come lunghezza massima del testo un solo carattere. Se invece il test precedente viene superato si chiede all'utente su quale directory vuole che l'unità virtuale

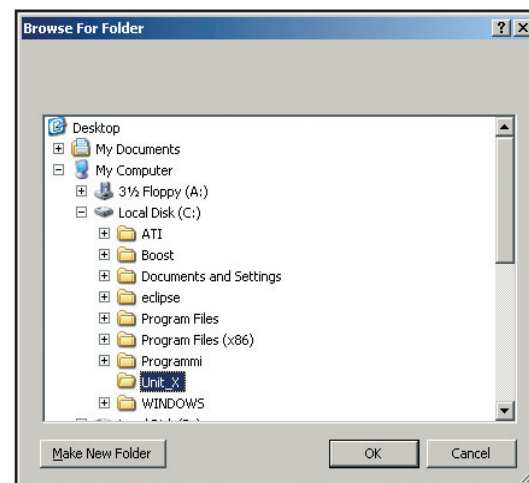



Figura 3: Selezione della cartella dove verrà montata l'unità virtuale.

venga montata come mostrato in **figura 3**.
A questo punto viene poi creata un'istanza della classe Process, e valorizzati opportunatamente alcuni campi in modo tale che sia in grado di eseguire correttamente il comando subst

```
    Process subst = new Process();
    subst.StartInfo.FileName = "subst";
    subst.StartInfo.Arguments =
                        textBoxLabel.Text + ": " +
               folderBrowserDialog1.SelectedPath;
```

il primo campo rappresenta semplicemente il nome del comando da eseguire, il secondo invece indica gli argomenti da passare al comando stesso. Come già illustrato, il comando subst per montare un'unità virtuale ne richiede l'etichetta e la directory di root-point. Una volta impostato bene l'oggetto è possibile invocare il metodo start per lanciare il comando. È importante notare che una volta invocato tale metodo si passa immediatamente all'istruzione successiva senza che il processo sia effettivamente terminato. Affinché ciò avvenga è quindi necessario invocare il metodo WaitForExit che attende il completamento del processo stesso.

```
    subst.Start();
    subst.WaitForExit();
```

SE AL POSTO DI WINDOWS C'È LINUX

L'applicazione è stata sviluppata per un ambiente Windows, infatti con Linux le cose sarebbero un po' più immediate. Ciò deriva dal fatto che per quest'ultimo tutto è un file, quindi anche un'immagine può essere montata e smontata come fosse un'unita, basta specificare il tipo di filesystem in questione. Ad esempio per un'immagine ISO 9660 basterà il comando:
mount myiso.iso /mnt/iso/ -t iso9660

Una volta terminato, tramite l'exit code, verifichiamo se il processo è andato a buon fine, in caso contrario lo segnaleremo con un messaggio d'errore. Infine aggiorneremo il controllo che visualizza le risorse del computer.

```
if (subst.ExitCode != 0)
    MessageBox.Show(this, "I cannot create
    this unit", "error!", MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
fileSystemListView1.RefreshView();
```

Il codice sottostante il secondo bottone, quello che serve per smontare l'unità virtuale, è concettualmente del tutto analogo a quello che si occupa del montaggio; naturalmente la stringa per valorizzare gli argomenti di start sarà diversa poiché in questo caso conterrà l'opzione /D

```
private void button1_Click(object sender, EventArgse)
    {
        //"Local Disk (X:)"
        try
        {
            string unit = getVirtualUnit();
            Process umount = new Process();
            umount.StartInfo.FileName = "subst";
            umount.StartInfo.Arguments = "/D " +unit;
            umount.Start();
            umount.WaitForExit();
            fileSystemListView1.RefreshView();
        }
        catch (Exception ex)
        {
            MessageBox.Show(this, ex.Message,
                    "error", MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
        }
    }
```

Un'altra variazione nella logica di questa operazione sta nella selezione dell'unità virtuale da smontare. Infatti è necessario che l'utente selezioni (dal controllo posto a sinistra) un'unità e non una directory o file qualsiasi; per questo è stato previsto il metodo getVirtualUnit() che si occupa proprio di accertarsi che sia stata effettivamente selezionata un'unità ed in caso affermativo ne restituisce l'etichetta.
Per adempiere a questo compito getVirtualUnit() fa uso di un'espressione regolare in modo tale da isolare solamente la lettera che indica l'etichetta dell'unità. Visto che non rientra negli scopi di questo articolo illustrare le regular expression mi limiterò esclusivamente a riportare il codice:

```
private string getVirtualUnit()
    {
        System.Text.RegularExpressions.Regex
                                regex = new
    System.Text.RegularExpressions.Regex(".*\\(([A-
                                        Z]:)\\)");
        string text =
            fileSystemListView1.SelectedItems[0].Text;
        if (!regex.IsMatch(text))
            throw new Exception("you must select a
                                        unit");
        return regex.Match(text).Groups[1].Value;
    }
```

ESTRARRE UN'IMMAGINE ISO

Una volta che siamo riusciti a montare e smontare un'unità virtuale non ci rimane altro che trovare un modo per estrarre un'immagine. Esistono vari formati di immagini, in quest'articolo focalizzeremo la nostra attenzione sul formato stabilito dall'organizzazione internazionale degli standard (ISO). Concettualmente però vedremo in seguito che se usassimo un altro tipo di formato non cambierà pressoché nulla ai fini della nostra applicazione. Giusto per avere un'idea di come sia definito tale standard spendiamo qualche frase per descrivere cosa sia un'immagine ISO 9660. Trattandosi appunto di uno standard, la seguente definizione è tratta da wikipedia:
ISO 9660 è il nome del *file system* standard per i *CD-ROM*, per i sistemi operativi *DOS/Windows*, *Macintosh* e *Unix*. Sviluppato nel 1987 dall'*International Organization for Standardization*. Le specifiche definiscono il formato della directory per un *CD-ROM* e un *CD-R* e prevedono tre livelli.

Il Livello 1 limita rigidamente la lunghezza dei nomi di *file* e cartelle al formato 8.3 del Dos (8 caratteri per il nome, 3 per l'eventuale estensione), con set di caratteri maiuscoli, numeri e underscore, nidificazione delle cartelle fino all'ottavo livello, con un massimo di 255 caratteri per percorso. I file devono essere scritti in settori contigui.

Il Livello 2 estende a 32 caratteri la lunghezza massima dei nomi di file e cartelle, ferme restando le restrizioni in termini di livello di nidificazione, lunghezza del percorso e scrittura in settori contigui.

Il Livello 3 abilita la scrittura di file in settori non contigui del supporto

L'esperienza fatta con il comando subst può tornarci utile anche in questo caso. Infatti se invece di un comando del nostro sistema operativo avessimo un file eseguibile da linea di comando che provvedesse all'estrazione dell'immagine il gioco sarebbe fatto. Come avrete sicuramente già intuito un programma che fa al caso nostro esiste e si chiama MagicISO, che

nella sua versione limitata, è scaricabile dal sito http://www.magiciso.com/download.htm. In particolare a noi non interessa l'applicazione nella sua interfaccia grafica, bensì due file che troverete nella directory d'installazione del programma: *miso.exe* e *misoh.dll*. Il primo è infatti proprio l'eseguibile invocando il quale potremmo estrarre l'immagine, mentre il secondo è una libreria necessaria all'eseguibile stesso. La prima cosa da fare è quindi copiare entrambi i file sul percorso di output del nostro progetto (per intenderci la directory bin dove viene creato l'exe della nostra applicazione), in modo tale che sappiamo che miso.exe sia sempre nella stessa directory della nostra applicazione. Va notato che la versione non registrata non può lavorare con immagini di dimensioni superiori ai 300MB.

UN WRAPPER PER UN PROCESSO

Estrarre un'immagine è sicuramente una procedura un po' più delicata di quanto non lo sia l'invocazione del comando subst visto precedentemente. Per questo motivo abbiamo deciso di creare una classe apposita che "incapsuli" il processo d'estrazione; per tale motivo la classe prende il nome di *ExtractProcess*. È sembrato opportuno concepire tale classe come un singleton, in modo tale che esista una ed una sola istanza che possa effettuare l'estrazione dell'immagine, quindi:

```
class ExtractProcess
  {
      private static ExtractProcess instance = null;
          .............

      public static ExtractProcess getInstance()
      {
          if (instance == null)
              instance = new ExtractProcess();
          return instance;
      }
```

Il costruttore sarà perciò privato e inizializzerà l'oggetto globale Process in questo modo:

```
  private ExtractProcess()
      {
          FileInfo app = new
   FileInfo(System.Windows.Forms.Application.Executabl
                                              ePath);
          string extractFileName =
             System.IO.Path.Combine(app.DirectoryName,
                                          "miso.exe");
          process = new Process();
          process.StartInfo.FileName =extractFileName;
          process.StartInfo.RedirectStandardError =true;
          process.StartInfo.RedirectStandardOutput =
                                                true;
          process.StartInfo.UseShellExecute = false;
          process.OutputDataReceived += new
             DataReceivedEventHandler(OutputHandler);
          process.ErrorDataReceived +=new
              DataReceivedEventHandler(ErrorHandler);
      }
```

La prima parte del codice è sostanzialmente analoga a quella vista in precedenza per il comando subst. La parte nuova è quella successiva, ossia:

```
          process.StartInfo.RedirectStandardError =true;
          process.StartInfo.RedirectStandardOutput =
                                                true;
          process.StartInfo.UseShellExecute = false;
          process.OutputDataReceived += new
             DataReceivedEventHandler(OutputHandler);
          process.ErrorDataReceived +=new
              DataReceivedEventHandler(ErrorHandler);
```

con queste righe reindirizziamo lo stream di output ed error verso la nostra classe, in particolare la "cattura" dei dati dai due stream è delegata rispettivamente ai metodi: OutputHandler e ErrorHandler. Tali dati sono memorizzati in uno StringBuilder:

```
class ExtractProcess
   {
         ..........
      private StringBuilder outputBuffer, errorBuffer;
         .........
private void OutputHandler(object sendingProcess,
                          DataReceivedEventArgs outLine)
      {
          outputBuffer.AppendLine(outLine.Data);
      }
  .....
    }
```

e poi esposta all'esterno come property read-only:

```
     public string OutputMessage
      {
          get
          {
              if (outputBuffer != null)
                  return outputBuffer.ToString();
              else
                  return string.Empty;
          }
      }
public int extract(string imageFile,string outputDir)
      {
          outputBuffer = new StringBuilder();
          errorBuffer = new StringBuilder();
          FileInfo image = new FileInfo(imageFile);
```

NOTA

Il controllo per l'esplorazione delle risorse del computer è reperibile all'indirizzo http://www.codeproject.com/cs/miscctrl/FileSystemListView.asp ed è facilmente integrabile con Visual Studio 2005.

```
        if (!image.Exists)
            throw new FileNotFoundException("file
                not found: " + imageFile, imageFile);
        DirectoryInfo output = new
                DirectoryInfo(outputDir);
        output.Create();
        string previousCurrentPath =
                System.Environment.CurrentDirectory;
        System.Environment.CurrentDirectory =
                outputDir;
        try
        {
            process.StartInfo.Arguments
              ="\""+imageFile+"\" "+ "-x " +outputDir;
            process.Start();
            process.BeginErrorReadLine();
            process.BeginOutputReadLine();
            process.WaitForExit();
            return process.ExitCode;
        }
        catch (Exception e)
        {
            throw e;
        }
        finally
        {
            System.Environment.CurrentDirectory =
                    previousCurrentPath;
            process.CancelErrorRead();
            process.CancelOutputRead();
        }
    }
```

In primo luogo verifichiamo che il file dell'immagine esista veramente, altrimenti lanciamo un'eccezione. Dopo di che valorizziamo gli argomenti del processo e lo lanciamo; subito dopo cominciamo a leggere lo standard output e quello error in modo che i metodi delegati precedentemente definiti possano catturarne le notifiche. Infine aspettiamo che il processo d'estrazione venga portato a termine.
Per tornare alla nostra GUI quindi il codice sottostante al terzo bottone, quello con l'etichetta "Estrai Immagine", sarà:

```
private void button2_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog(this) !=
                DialogResult.OK)
            return;
        ExtractProcess p =
                ExtractProcess.getInstance();
        try
        {
            string unit = getVirtualUnit();
            p.extract(openFileDialog1.FileName,unit);
            MessageBox.Show(this, "Immage
             extracted sucessfully into " + unit + " unit",
                string.Empty, MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show(this, ex.Message,
                "error", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
```

I tre passi che l'utente deve compiere sono riportate in **figura 4-5-6**.

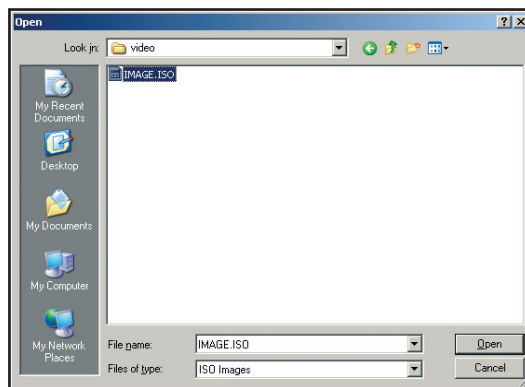



Figura 4: Selezione dell'immagine che verrà estratta.

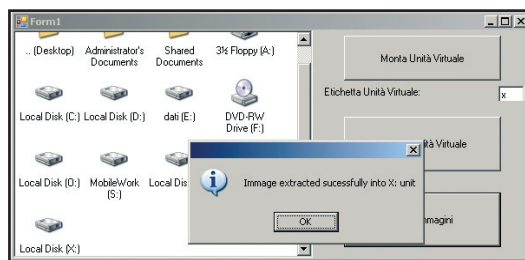



Figura 5: Estrazione dell'immagine sull'unità virtuale

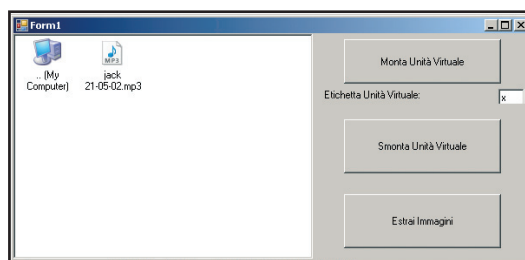



Figura 6: Esplorazione dell'unità virtuale per verificare che l'immagine sia stata estratta correttamente.

CONCLUSIONI

In questo articolo abbiamo sviluppato un'applicazione che può risultare nella pratica spesso utile in diverse occasioni. Inoltre è stata colta l'occasione per capire come all'interno del framework .NET si possa avere il controllo di processi esterni ad esso.

Andrea Galeazzi

L'AUTORE

Laureato in ingegneria elettronica presso l'università Politecnica delle Marche, lavora presso il reparto R&D della Korg S.p.A. Nei limiti della disponibilità di tempo risponde all'indirizzo andrea-galeazzi@fsfe.org

DAI UNA CACHE AL TUO SOFTWARE

LE APPLICAZIONI NON SONO MAI ABBASTANZA VELOCI. LE TECNICHE DI CACHING SONO UNO DEI METODI PRINCIPALI PER AUMENTARNE LE PERFORMANCE. IMPARIAMO AD UTILIZZARE IL CACHING APPLICATION BLOCK CON MICROSOFT VISUAL STUDIO

Qualsiasi tipo di applicazione realizziamo per lavoro o per diletto, necessiterà prima o poi in alcune sue parti di utilizzare qualche tecnica per ottimizzare la gestione dei dati e ridurre il tempo di attesa dell'utente durante elaborazioni o query particolarmente complesse. Per quanto riguarda le applicazioni Web il .NET Framework ci viene abbondantemente in aiuto fornendoci diverse possibilità per implementare tecniche di caching. Il namespace *System.Web.Cache* ci fornisce infatti tutte le classi necessarie per utilizzare la cache nelle nostre applicazioni Web e nei Web Services.

La cache di ASP.NET però, sebbene possa essere utilizzata anche in applicazioni non Web, fornisce il meglio di se solo in questo ambiente ma soprattutto non essendo stata progettata specificamente per lavorare fuori da questo ambiente, non è supportata da Microsoft se utilizzata in applicazioni non Web. Per fornire uno strumento più generico, completo ed utilizzabile in qualsiasi tipo di applicazione, Microsoft ha quindi sviluppato il *Caching Application Block (CAB)*, ovvero una soluzione di caching totalmente gratuita e robusta inclusa nel più ampio pacchetto di soluzioni denominato *Enterprise Library*, giunto al momento in cui scriviamo alla versione *3.1* di *Maggio 2007*.

Prima del rilascio di questa libreria e del *Caching Application Block*, gli sviluppatori dovevano creare di proprio pugno i metodi per la gestione della cache con il rischio di introdurre nuovi colli di bottiglia o nuovi bug e con tutte le complicazioni che una implementazione del genere comporta. Con il *Caching Application Block* è possibile gestire la cache in qualsiasi livello della nostra applicazione, quindi nello strato dati, nello strato business od anche nello strato di presentazione (considerando la classica architettura a tre livelli). Vediamo quindi di applicare praticamente questa libreria andando a sviluppare una semplice applicazione demo che faccia uso del CAB nella lettura di file di testo e nel recupero di dati da database. Vedremo la notevole differenza che ci sarà quando utilizzeremo o meno la cache per recuperare i dati o per leggere i file.

INSTALLAZIONE

Innanzi tutto recuperiamo la *Enterprise Library* effettuando il download dal sito Microsoft ed installiamola sulla nostra macchina. Il processo di installazione non richiede particolari accorgimenti, è sufficiente infatti specificare solo il percorso di installazione degli assembly della libreria e successivamente il percorso dei sorgenti e dei progetti Quick Start. L'ultimo passo del processo di installazione consiste nella compilazione di tutti i sorgenti e richiederà alcuni secondi durante i quali sarà visualizzata una finestra che mostrerà i nomi dei file sorgente mentre vengono compilati. Se non si sono modificati i percorsi di installazione, gli assembly che dovremo referenziare nel nostro progetto saranno presenti nel percorso:

```
<disco>:\Program Files\Microsoft Enterprise Library
                                    3.1 - May 2007\Bin
```

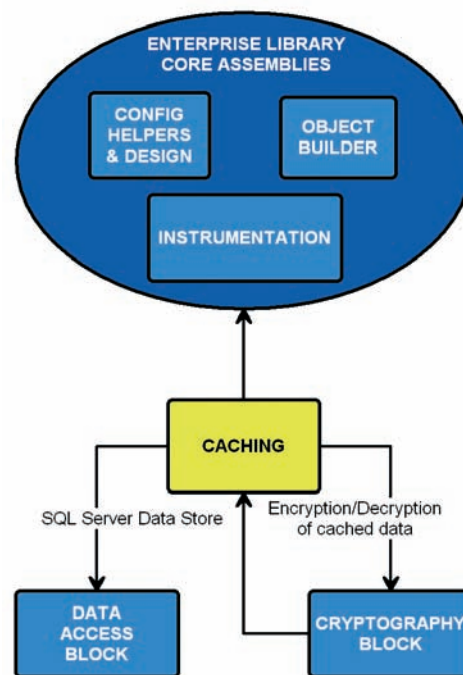



***Figura 1:** Diagramma delle dipendenze di Caching Application Block*

In particolare, tra i diversi assembly dell'intera *Enterprise Library*, per utilizzare le funzionalità di base del *Caching Application Block*, dovremo referenziare nel nostro progetto solo: *Microsoft.Practices.EnterpriseLibrary.Common.dll* e *Microsoft.Practices.EnterpriseLibrary.Caching.dll*.
Di default il *Caching Application Block* usa come repository della cache la memoria di sistema ma è possibile anche scegliere di utilizzare un database SQL Server impostando opportunamente alcuni parametri. Per utilizzare un database SQL però è necessario referenziare anche gli assembly del *Data Access Application Block*. E se dovessimo avere la necessità di criptare i dati della cache per questioni di sicurezza in particolari applicazioni in cui la riservatezza dei dati è importante, è possibile anche utilizzare la criptazione / decriptazione dei dati tramite l'uso del *Cryptography Application Block*.

IMPLEMENTAZIONE

Avviamo Visual Studio e creiamo un nuovo progetto Windows Application da *File -> New Project...*, scegliamo quindi il template *Windows Application*, assicuriamoci che come Language ci sia Visual C# e premiamo OK. Come prima operazione, referenziamo i due assembly necessari all'utilizzo del Caching Application Block, ovvero *Microsoft.Practices.EnterpriseLibrary.Common.dll* e *Microsoft.Practices.EnterpriseLibrary.Caching.dll* dalla cartella *<disco>:\Program Files\Microsoft Enterprise Library 3.1 - May 2007\Bin*. Aggiungiamo anche un file di configurazione nel quale dovremo inserire alcune voci relative alla configurazione del nostro application block. Clicchiamo quindi con il tasto destro del mouse sul progetto, scegliamo *Add -> New Item...*, selezioniamo il tipo di file *Application Configuration File* e quindi premiamo *Add*.
A questo punto nel file *App.config* appena aggiunto copiamo le seguenti chiavi di configurazione all'interno del nodo *configuration*:

```
<configSections>
    <section name="cachingConfiguration"
     type="Microsoft.Practices.EnterpriseLibrary.
     Caching.Configuration.CacheManagerSettings,
     Microsoft.Practices.EnterpriseLibrary.Caching" />
</configSections>
<cachingConfiguration
     defaultCacheManager="Default Cache Manager">
    <backingStores>
     <add name="inMemory"
     type="Microsoft.Practices.EnterpriseLibrary.
Caching.BackingStoreImplementations.
                                    NullBackingStore,
      Microsoft.Practices.EnterpriseLibrary.Caching" />
    </backingStores>
    <cacheManagers>
     <add name="Default Cache Manager"
      expirationPollFrequencyInSeconds="60"
maximumElementsInCacheBeforeScavenging="1000"
      numberToRemoveWhenScavenging="10"
      backingStoreName="inMemory" />
     </cacheManagers>
</cachingConfiguration>
```

Con queste chiavi di configurazione abbiamo definito una sezione di configurazione per il CAB nella quale abbiamo specificato una serie di parametri tra i quali il *backingStoreName* ovvero il nome della chiave che definisce il repository della nostra cache. Il backing store specificato (*inMemory*) fa riferimento al namespace:

```
Microsoft.Practices.EnterpriseLibrary.Caching.Backing
                                StoreImplementations
```

e nello specifico alla classe NullBackingStore ovvero all'implementazione della cache in memoria. Le altre implementazioni specificabili sono:

- **IsolatedStorageBackingStore**
  Gli item sono immagazzinati nello storage specifico dell'application domain, ovvero una particolare cartella del disco riservata all'applicazione
- **DataBackingStore**
  Utilizza un database come backing store dei dati ed utilizza a sua volta il *Data Access Application Block*

L'implementazione *NullBackingStore* a cui faremo riferimento in questo articolo è l'implementazione in memoria della cache e per tale motivo è una implementazione che non persiste gli item. Questo significa che per esempio a differenza dell'implementazione *DataBackingStore*, una volta terminata l'applicazione, gli item in cache saranno eliminati. L'applicazione è composta da tre *Windows Form*. Un form è utilizzato come entry point dell'applicazione e visualizza due tasti attraverso i quali è possibile scegliere la demo da avviare, mentre gli altri due form contengono rispettivamente la demo relativa alla lettura di file di testo e quella relativa alla lettura di dati dal database. Per la demo facente uso del database utilizziamo il ben noto database *Northwind* fornitoci dalla Microsoft proprio per l'utilizzo in applicazioni di prova. Il form *frmFileViewer* ci mostra come il caricamento di un file di testo di ragguardevoli dimensioni può essere accelerato tramite l'utilizzo della cache mentre il form *frmTableViewer* dimostrerà quanto le performance di un'applicazione facente uso di database possano essere migliorate tramite l'utilizzo di CAB.

NOTA

DOVE TROVARE LA LIBRERIA ENTERPRISE LIBRARY 3.1 MAY 2007

La Enterprise Library 3.1 May 2007 è disponibile al seguente indirizzo:
http://www.microsoft.com/downloads/details.aspx?FamilyID=4c557c63-708f-4280-8f0c-637481c31718&displaylang=en

UTILIZZO DELLA CACHE NELLA LETTURA DI FILE

Come abbiamo detto poco sopra, partiamo innanzi tutto con lo specificare due using che ci permetteranno di utilizzare agevolmente i *namespace* del CAB:

```
using Microsoft.Practices.EnterpriseLibrary.Common;
using Microsoft.Practices.EnterpriseLibrary.Caching;
```

dopo di che creiamo nel costruttore del form una istanza di classe del *CacheManager*, ovvero l'oggetto principale attraverso il quale potremo utilizzare la cache messa a disposizione dal *Caching Application Block*:

```
private CacheManager _manager;

public frmFileViewer()
{
      InitializeComponent();

      // Creo l'oggetto CacheManager per la gestione
                                              della Cache
      _manager = CacheFactory.GetCacheManager();
}
```

Il metodo *GetCacheManager* restituisce il cache manager di default definito nel file di configurazione (nel nostro caso "*Default Cache Manager*") ma se avessimo diversi cache manager potremmo specificarne il nome passando quest'ultimo come parametro:

```
_manager = CacheFactory.GetCacheManager("Other
                                   Cache Manager");
```

A questo punto vediamo il metodo che salva le righe del file selezionato nella cache:

```
private void StoreFileInCache(string fileName)
{
   if (!_manager.Contains(txtFileName.Text))
   {
      int lineNumber = 0;
      string[] fileContent =
                    File.ReadAllLines(txtFileName.Text);
      string[] cachedContent = new
                         string[fileContent.Length];

      foreach (string line in fileContent)
      {
       lineNumber++;
       cachedContent[lineNumber-1] =
   lineNumber.ToString().PadLeft(10, '0') + " " + line;
      }

      _manager.Add(txtFileName.Text,
                                      cachedContent);
   }
}
```

Qui verifichiamo che nella cache sia presente un item avente come chiave il nome del file stesso (ma potevamo utilizzare una qualsiasi altra chiave di tipo *string*) e se non presente recuperiamo dal file selezionato tutte le righe attraverso il metodo statico *ReadAllLines* della classe *File* e poi dopo aver aggiunto ad ogni riga del file il numero di riga inseriamo l'array di *string* così ottenuto nella cache dandogli come chiave il nome stesso del file.
Ora vediamo i due metodi che caricano il file in una *ListBox*, l'uno ricaricando nuovamente il file da disco, l'altro recuperando il file dalla cache. Qui il vantaggio nell'uso della cache sarà apprezzabile solo con file di dimensioni elevate in quanto in realtà la lettura da disco utilizza già un buffer di lettura che mette in cache alcune porzioni dei file. Ma soprattutto l'uso della cache è indicato nel momento in cui i dati (in questo caso le righe del file) sono oggetto di elaborazioni anche pesanti che richiedono quindi molto tempo di CPU e che grazie all'uso della cache possono essere eseguite una volta sola anziché ad ogni lettura del file. Un altro scenario in cui il CAB è molto indicato è la richiesta di dati da Web Service i quali possono implicare tempi di accesso considerevoli che la cache permette di ridurre notevolmente.

```
private cmdLoadWithoutCache_Click(sender, e)
{
   if (txtFileName.Text != "")
   {
      lbFileContent.Items.Clear();
      int lineNumber = 0;
      int startTime = .TickCount;

      string[] fileContent =
                        .ReadAllLines(txtFileName.Text);

      foreach (line fileContent)
      {
       lineNumber++;
        lbFileContent.Items.Add(
   lineNumber.ToString().PadLeft(10, '0') + " "+ line);
      }

       txtLoadingtime.Text =
            Convert.ToString(.TickCount - startTime);
   }
}
```

Con il gestore dell'evento click del tasto *cmdLoadWithoutCache* leggiamo nuovamente le righe dal file selezionato, aggiungiamo i numeri di riga e ripopoliamo la *ListBox* con le righe del file. Quin-

NOTA

GLI ALTRI APPLICATION BLOCK DELLA ENTERPRISE LIBRARY

La Enterprise Library oltre al Caching Application Block, contiene anche altri utili application block: Cryptography Application Block, Data Access Application Block, Exception Handling Application Block, Logging Application Block, Policy Injection Application Block, Security Application Block e Validation Application Block

di senza utilizzare la cache.

```
private void cmdLoadWithCache_Click(object sender,
                                     EventArgs e)
{
   if (txtFileName.Text != "")
   {
      if (_manager.Contains(txtFileName.Text))
      {
         lbFileContent.Items.Clear();

         int startTime = Environment.TickCount;
         string[] rows =
                  (string[])_manager[txtFileName.Text];

         foreach (string row in rows)
         {
            lbFileContent.Items.Add( row);
         }

         txtLoadingtime.Text =
            Convert.ToString(Environment.TickCount -
                                          startTime);
      }
   }
}
```

Con questo metodo invece recuperiamo le righe del file dalla cache e le visualizziamo nella *ListBox* senza rileggerle dal disco e senza dover aggiungere nuovamente i numeri di riga.
In entrambi i casi, grazie all'utilizzo della proprietà *Environment.TickCount* che tiene conto dei millisecondi trascorsi dall'avvio del sistema, abbiamo calcolato il tempo dell'operazione svolta visualizzandolo in un'apposita casella di testo.
Ecco quindi i tempi di lettura visualizzati nell'applicazione con l'utilizzo della cache:

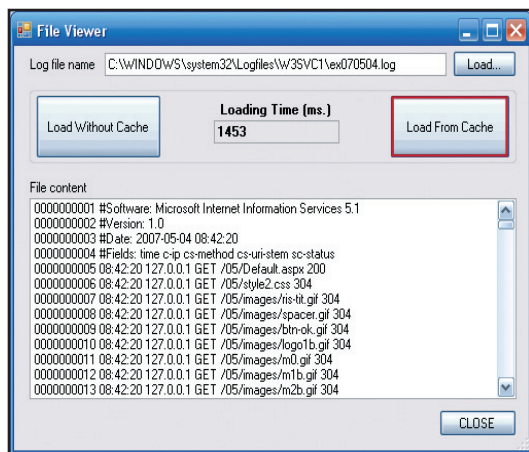



Figura 2: Lettura file di testo dalla cache

e senza l'utilizzo della cache, ovvero leggendo il file direttamente da disco:

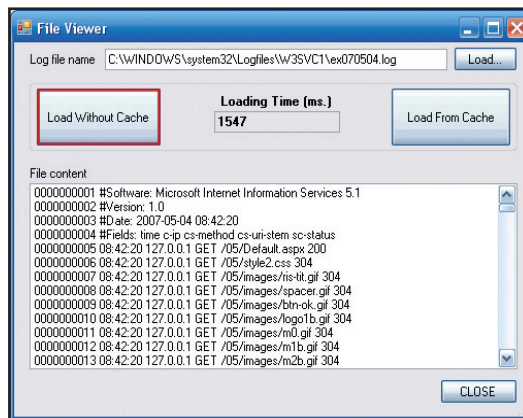



Figura 3: Lettura file di testo da disco

Come possiamo notare la differenza, seppur minima in questa demo, è comunque rilevabile.

UTILIZZO DELLA CACHE NELLA LETTURA DI DATI DA DATABASE

Passiamo quindi al secondo form, quello facente uso del database. In questo form abbiamo una *GridView* che visualizzerà il contenuto della tabella *Order_Details* del database *Northwind*. Anche in questo caso abbiamo un metodo che salva i dati in cache dopo averli letti dal database:

```
private StoreDataInCache()
{
this.order_DetailsTableAdapter.Fill(_northwindDataSet
                                   .Order_Details);
      _manager.Add("OrderDetails",
                  _northwindDataSet.Order_Details);
}
```

Qui però in cache salviamo una intera *DataTable* riempita con i dati del database *Northwind* ed in particolare della tabella *Order_Details*, che utilizziamo in quanto contenente un numero rilevante di record che fanno al caso nostro. Il metodo *StoreDataInCache* viene richiamato per comodità in fase di *Load* del form. Così come nell'esempio precedente abbiamo qui due metodi per il caricamento dei dati, uno che utilizza la cache ed un altro che invece non ne fa uso:

```
private void cmdLoadWithoutCache_Click(object
                          sender, EventArgs e)
{
      int startTime = Environment.TickCount;

this.order_DetailsTableAdapter.Fill(_northwindDataSet
                                   .Order_Details);
      BindGridView(_northwindDataSet.Order_Details);
      txtLoadingtime.Text =
```

NOTA

REFERENZIARE DLL NEI PROGETTI

In Visual Studio per referenziare un assembly in un progetto è sufficiente cliccare con il tasto destro del mouse sul nome del progetto, scegliere la voce Add Reference... , cliccare sul tab Browse e selezionare dal disco rigido la DLL che s'intende referenziare nel progetto.

NOTA

IL DATABASE NORTHWIND

Il database Northwind è il tipico database di test pensato dalla Microsoft per essere utilizzato nelle applicazioni demo. E' disponibile un pacchetto d'installazione contenente il database Northwind e il database Pubs (altro db di demo) al seguente indirizzo:
http://www.microsoft.com/downloads/details.aspx?FamilyID=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en

```
                    Convert.ToString(Environment.TickCount -
                                                   startTime);
}
```

Con questo metodoeffettuiamo la *Fill* della *DataTableOrder_Detailse* quindi la mettiamo in binding con la *GridView* per mostrarne il contenuto.

```
private cmdLoadWithCache_Click(sender, e)
{
    int startTime = .TickCount;
BindGridView((NorthwindDataSet._DetailsDataTable)_
                         manager["OrderDetails"]);
     txtLoadingtime.Text =
              Convert.ToString(.TickCount - startTime);
}
```

In questo metodo invece mettiamo in binding la *GridView* direttamente con la *DataTable* contenuta nella cache e riempita in precedenza tramite il metodo *StoreDataInCache*.

```
private void BindGridView(
        NorthwindDataSet.Order_DetailsDataTable dt)
{
    // Inserisce un valore casuale nella colonna
                          UnitPrice della prima riga
    // per rendere visibile l'aggiornamento della
                                           GridView
    Random rnd = new Random();
    dt[0].UnitPrice = (decimal)rnd.Next();

    gvOrderDetails.DataSource = dt;
}
```

Infine troviamo il metodo *BindGridView* nel quale mettiamo la *GridView* in binding con la DataTable ed inseriamo nella colonna *UnitPrice* della prima riga un valore casuale al solo scopo di renderci visibile l'aggiornamento della *GridView* ad ogni pressione dei due tasti di caricamento.
In questo secondo esempio la differenza di prestazioni sarà più evidente rispetto all'esempio con i file di testo in quanto è sicuramente molto più oneroso per la macchina caricare una intera tabella contenente centinaia di record dal database piuttosto che dalla memoria attraverso la cache.
Vediamo allora i tempi di lettura visualizzati nell'applicazione con l'utilizzo della cache:

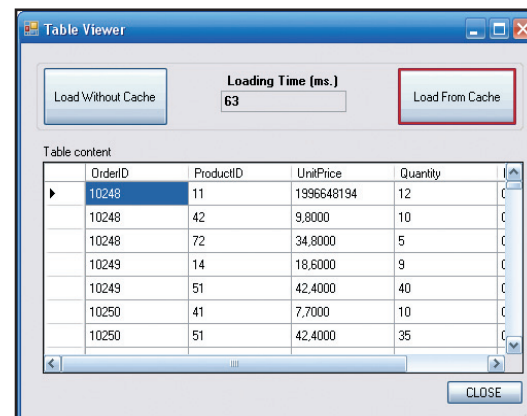



Figura 4: Lettura dati dalla cache

e senza l'utilizzo della cache, ovvero leggendo i dati dal database:

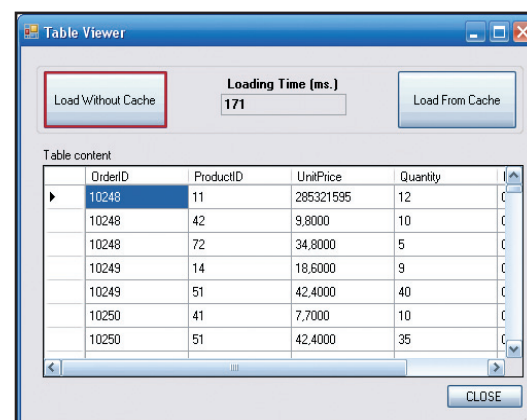



Figura 5: Lettura dati direttamente dal database

L'OGGETTO CACHEMANAGER

Dopo aver visto questo esempio pratico di utilizzo della cache del CAB passiamo quindi ad analizzare l'oggetto *CacheManager*, fulcro e cuore dell'intero application block. Questa classe non è una classe molto complessa, infatti contiene (nell'implementazione *NullBackingStore*) solo sei metodi significativi e tutti molto semplici da usare. La sua potenza infatti sta proprio in questo. Il CAB nasconde allo sviluppatore la complessa logica di gestione del caching fornendo metodi semplici ed auto esplicativi. Ecco un elenco riassuntivo dei metodi e della loro funzione:

- **Add**
  Aggiunge un nuovo item alla cache. Se in cache esiste già un item con la stessa chiave di quello che si sta inserendo, l'item esistente viene rimosso e

COSA SONO GLI APPLICATION BLOCK

Gli application block di cui in questo articolo utilizzeremo quello dedicato alla gestione del caching, sono componenti riutilizzabili ed estendibili (in quanto è a disposizione anche il codice sorgente) che forniscono una guida per la soluzione dei più comuni problemi di programmazione. La Enterprise Library è una libreria che raccoglie tutti gli application block aggiungendovi inoltre documentazione ed esempi di utilizzo utili per famigliarizzare con ogni application block ed imparare velocemente ad utilizzarli.

sostituito con il nuovo.

- **Contains**
  Data la chiave di un item indica se questo è già presente o meno in cache
- **Count**
  Restituisce il numero totale di item della cache
- **Flush**
  Rimuove tutti gli item dalla cache
- **GetData**
  Ritorna uno specifico item data la sua chiave. Gli item vengono sempre restituiti di tipo object. Deve quindi essere effettuato un cast al tipo corretto dell'item. Un item può essere inoltre recuperato anche direttamente senza l'utilizzo di GetData attraverso il metodo implicito *Item. Es. object item = _manager["chiave"];*
- **Remove**
  Rimuove uno specifico item data la sua chiave

Vediamo in particolare il metodo *Add* che fornisce due overload. Il metodo *Add* come abbiamo detto permette l'inserimento di un item in cache e nella sua versione più semplificata accetta come parametri semplicemente la chiave dell'item e l'item stesso sotto forma di *object*. Il metodo *Add* accetta però, oltre alla chiave e all'item, anche altri parametri nel suo secondo overload:

- **scavengingPriority**
  Specifica la priorità dell'item nel momento in cui la cache viene liberata. Questo parametro è di tipo *CacheItemPriority* e può assumere i valori: *High, Low, None, Normal, NotRemovable*. Il valore di default (se non specificato) è *Normal*.
- **refreshAction**
  Questo parametro di tipo *ICacheItemRefreshAction*, specifica una classe che viene richiamata ogni qual volta un item scade e permette di intraprendere una qualsiasi azione per aggiornare l'item prima che questo scada.
- **espirations**
  Questi parametri permettono di specificare una serie di classi derivanti da *ICacheItemExpiration* implementanti delle regole che definiscono come e quando un item può scadere. Di default, se non specificato diversamente, gli item non scadono.

Le classi che implementano *ICacheItemRefreshAction* e *ICacheItemExpiration* devono essere serializzabili.

ENTERPRISE LIBRARY CONFIGURATION

All'inizio di questo articolo abbiamo creato un file di configurazione in cui abbiamo inserito una serie di chiavi necessarie per definire il comportamento del CAB. Queste stesse chiavi possono essere gestite in una maniera più amichevole attraverso l'applicazione *Enterprise Library Configuration* che possiamo trovare, dopo aver installato correttamente la *Enterprise Library*, nel menu:

Start -> All programs -> Microsoft patterns & practices -> Enterprise Library 3.1 - May 2007 -> Enterprise Library Configuration

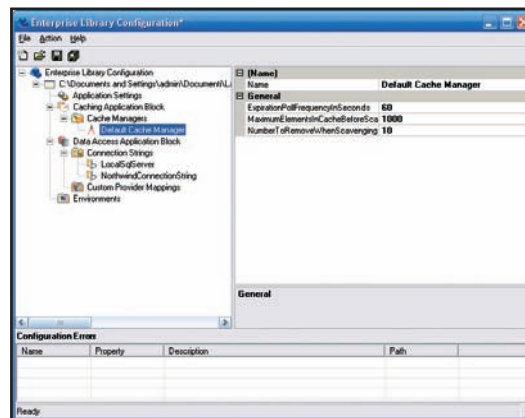



Figura 6: Enterprise Library Configuration

Attraverso questa comoda interfaccia potremo configurare tutti gli application block della *Enterprise Library*, tra cui quindi anche il *Caching Application Block*, semplicemente aprendo il file *App.config* del nostro progetto demo dal menu *File -> Open Application* e modificando opportunamente i parametri di configurazione.

CONCLUSIONI

Il caching è uno degli aspetti fondamentali dello sviluppo software ed una implementazione solida di queste tecniche nelle nostre applicazioni può essere spesso determinante nella riuscita delle stesse. La gestione della cache dei dati è un problema molto noto agli sviluppatori web che difatti utilizzano praticamente sempre tecniche simili per velocizzare le proprie web application. Ma anche nello sviluppo desktop si deve avere particolare attenzione verso i tempi di risposta degli applicativi perché se è vero che i computer attuali sono sempre più potenti è anche vero che le applicazioni attuali devono gestire moli di dati sempre maggiori ed una corretta gestione dei tempi di risposta è necessaria se non fondamentale per garantire una buona usabilità da parte dell'utente. Come abbiamo visto l'utilizzo del *Caching Application Block* è abbastanza semplice e quindi è nostro compito solo individuare i punti giusti dove utilizzare la cache per rendere sempre più performanti le nostre applicazioni.

Gianni Malanga

L'AUTORE

Lavora da più di 10 anni con tecnologie Microsoft in particolare nel campo dello sviluppo Web. Sviluppa in .NET ormai da alcuni anni e ha svolto diverse docenze per corsi di formazione professionali. Dopo aver lavorato alle dipendenze di importanti realtà locali, attualmente ha intrapreso la libera professione costituendo la ditta individuale Kaone Consulting che si occupa di consulenza informatica per PMI su tutto il territorio nazionale. Gli si può scrivere all'indirizzo kaone@email.it e il suo blog è disponibile su: http://thekaone.blogspot.com

XPS: L'ALTERNATIVA MICROSOFT AL PDF

OLTRE ALLE MIGLIAIA DI FUNZIONALITÀ RIVOLTE ALL'ASPETTO GRAFICO, WINDOWS PRESENTATION FOUNDATION, CI METTE A DISPOSIZIONE UNA SERIE DI SERVIZI RIVOLTI ALLA GESTIONE DEI DOCUMENTI. INTEGRIAMO IL NUOVO XPS NEL NOSTRO SOFTWARE

Nella maggior parte delle demo viste su Windows Presentation Foundation assistiamo a veri e propri spettacoli scenografici con interfacce molto accattivanti ed innovative. Effettivamente la potenzialità principale di Windows Presentation Foundation è proprio quella di mettere a disposizione classi per semplificare la creazione di interfacce sempre più vicine a quello che ormai l'utente finale si sta abituando con la diffusione del web.
Tuttavia, Windows Presentation Foundation, mette a disposizione avanzate funzionalità per semplificare la creazione di contenuti rivolti alla lettura introducendo, quindi, nuove funzionalità per la gestione documentale attraverso classi utili alla formattazione, impaginazione e per la creazione di pacchetti nel nuovo formato *XML Paper Specification (XPS)* sviluppato da Microsoft come alternativa al già conosciuto PDF creato da Adobe.
Diamo, innanzitutto, uno sguardo alla gestione dei documenti in Windows Presentation Foundation.

Conoscenze richieste
Conoscenze minime del .NET framework 2.0, conoscenze di base di Windows Presentation Foundation, conoscenze dei concetti di base dei linguaggi di markup

Software
Visual Studio 2005 o superiore; .NET 3 Runtime; Visual Studio Extensions for Windows Presentation Foundation

Impegno

Tempo di realizzazione

DOCUMENTI CON WPF

Come già accennato, Windows Presentation Foundation, mette a disposizione avanzate funzionalità per semplificare la creazione di contenuti rivolti alla lettura, quindi documenti. In generale, quando abbiamo bisogno di visualizzare del testo come semplici etichette o titoli nelle nostre *Window*, utilizziamo *Label* o il nuovo controllo *TextBlock* ma, in casi di veri e propri documenti utilizzeremo le classi messe a disposizione ad Windows Presentation Foundation.
Abbiamo, innanzitutto, due categorie di documenti a disposizione denominati "*Fixed Document*" e "*Flow Document*". Come è già facilmente intuibile dai nomi, i FixedDocument sono documenti con una struttura fissa a prescindere, quindi, dal tipo di supporto di visualizzazione che può essere un monitor piuttosto che una stampante. Con la classe FlowDocument, invece, possiamo definire dei documenti caratterizzati dal fatto che questi si adattano in maniera ottimale al supporto in cui vengono visualizzati.

UN FLOW-DOCUMENT PER UN ARTICOLO DI GIORNALE

L'utilizzo di un documento di tipo FlowDocument può essere utilizzo per creare delle pagine di un giornale che può essere visualizzato a video o stampato e non ha bisogno di una struttura fissa come lo dovrebbe essere, ad esempio, per una fattura, ma deve adattarsi al device su cui viene visualizzato per dare una maggiore comodità nella lettura all'utente.
Un banale esempio di definizione di un FlowDocument è il seguente.

```
<FlowDocument
xmlns="http://schemas.microsoft.com/winfx/2006/xaml
                                      /presentation"
  ColumnWidth="200" FontSize="16"
                              FontFamily="Georgia"
  >
 <Paragraph FontSize="22" FontWeight="Bold">
  DOCUMENTI IN WPF
 </Paragraph>
 <Paragraph>
   Nella maggior parte delle demo viste su Windows
   Presentation Foundation assistiamo a veri e propri
                                              [...]
 </Paragraph>
 <Paragraph>
 Tuttavia, Windows Presentation Foundation, mette a
 disposizione avanzate funzionalità per semplificare la
                                              [...]
 </Paragraph>
 <Paragraph
   Diamo, innanzitutto, uno sguardo alla gestione dei
     documenti in Windows Presentation Foundation.
```

```
  </Paragraph>
  <Paragraph>
          Come già accennato, Windows Presentation
          Foundation, mette a disposizione avanzate
                                               [...]

  </Paragraph>
</FlowDocument>
```

Questo è un semplice documento che, come vedete, utilizza come elemento root un FlowDocument mentre, all'interno troviamo diversi paragrafi definiti attraverso l'elemento *Paragraph*. In **Fig.1** vediamo il nostro documento visualizzato all'interno dell'apposito viewer messo a disposizione dal framework con le classiche funzionalità di zooming, navigazione tra pagine, eccetera.

In questo caso il viewer viene visualizzato automaticamente visto che la Window di avvio del progetto è proprio un *FlowDocument*. Infatti non è possibile utilizzare un *FlowDocument* direttamente all'interno di una normale o meglio è possibile ma non direttamente. Un *FlowDocument* infatti non viene renderizzato in una Window ma è necessario utilizzare un *FlowDocumentReader* che contenga il *FlowDocument* da visualizzare.

```
<Window x:Class="FlowSample.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml
                                        /presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
                                                ml"
    Title="FlowSample" Height="300" Width="300"
    >
    <Grid>
      <FlowDocumentReader>
        <FlowDocument>
          <!-- Definizione del documento -->
        </FlowDocument>
      </FlowDocumentReader>
    </Grid>
</Window>
```

Altra caratteristica del documento appena creato è il fatto che, ridimensionando la Window, i paragrafi vengono sempre adattati alle nuove dimensioni visto che questo è stato definito, appunto, come FlowDocument. Purtroppo l'immagine statica dell'articolo non rende bene l'idea e per questo vi consiglio di provare l'esempio voi stessi.

Nell'esempio, inoltre, vengono utilizzati gli oggetti *Paragraph* che in Windows Presentation Foundation fanno parte dei cosiddetti *Blocks* ossia degli oggetti che vengono utilizzati per definire dei gruppi di testo e che, internamente, sono classi che derivano dalla classe *System.Windows.Documents.Block*.

Windows Presentation Foundation mette a disposizione cinque tipi di *Blocks*:

- *Paragraph* – contiene una collection di Inlines che sono il contenuto del paragrafo. Da codice XAML è possibile indicare semplicemente la stringa di testo oppure utilizzando gli oggetti Run che aggiungono alla collection di Inlines un elemento.
- *Section* – utile per raggruppare delle sezioni che devono avere le stesse caratteristiche.
- *List* – genera un elenco numerato, puntato o cumunque una semplice lista. Tra le varie possibilità offerte c'è quella di personalizzare l'aspetto dei puntatori.
- *Table* – permette l'organizzazione dei contenuti in righe e colonne.
- *BlockUIContainer* – può contenere alcuni controlli WPF come Button, Image, contenuti 3D o MediaElement.

Tornando all'esempio precedente è possibile, quindi, utilizzando un *BlockUIContainer* inserire delle immagini.

```
<FlowDocument
xmlns="http://schemas.microsoft.com/winfx/2006/xaml
                                        /presentation"
    ColumnWidth="200" FontSize="16"
                                  FontFamily="Georgia"
    >
  <Paragraph FontSize="22" FontWeight="Bold">
    DOCUMENTI IN WPF
  </Paragraph
  <BlockUIContainer>
    <Image Source="NETFw.jpg"/>
  </BlockUIContainer
  <Paragraph>

Nella maggior parte delle demo viste su Windows
     Presentation Foundation assistiamo a veri e propri
          spettacoli scenografici con interfacce molto
                                               [...]
```

Figura 1: *Il nostro articolo visualizzato con WPF*

```
  </Paragraph>
  <Paragraph>

Tuttavia, Windows Presentation Foundation, mette a
  disposizione avanzate funzionalità per semplificare la
                                                  [...]
  </Paragraph>
  <Paragraph>
     Diamo, innanzitutto, uno sguardo alla gestione dei
        documenti in Windows Presentation Foundation.
                                                  [...]
  </Paragraph>
  <Paragraph>
          Come già accennato, Windows Presentation
           Foundation, mette a disposizione avanzate
                                                  [...]
  </Paragraph>
</FlowDocument>
```

Abbiamo accennato, a proposito del Block *Paragraph*, alla collection di elementi *Inlines* che possono essere inseriti in un, appunto, Paragraph per far si che il suo contenuto sia più interessante di un semplice testo piatto. Per aggiungere elementi alla collection *Inlines*, occorre utilizzare la proprietà *Run* che aggiunge, quindi, una stringa di testo al paragrafo.

```
<Paragraph>
  <Run>Prova di aggiunta elemento Inlines</Run>
  <Run>Secondo elemento Inlines</Run>
</Paragraph>
```

E' possibile, per ogni Run, specificare caratteristiche diverse come ad esempio il Font, il colore, lo stile.

```
<Paragraph>
  <Run FontSize="20">Prova di aggiunta elemento
                                    Inlines</Run>
  <Run FontWeight="Bold">Secondo elemento
                                    Inlines</Run>
</Paragraph>
```

È possibile specificare un interruzione di pagina e, per fare questo, occorre impostare, sul primo paragrafo della nuova pagina, la proprietà *BreakPageBefore* a *True*. In questo modo, verrà creata una nuova pagina il cui primo paragrafo sarà quello in cui abbiamo specificato *BreakPageBefore*.

IL FORMATO XPS

Con l'avvento di Windows Vista e del .NET Framework 3.0, Microsoft, ha sviluppato un nuovo formato documentale chiamato, appunto, *XPS*. Questo formato si basa sulle specifiche *Open Packaging Conventions* e alle *Open XML Markup Compatibility* che descrivono come creare un pacchetto di contenuti in maniera standardizzata.
Un documento *XPS* è, sostanzialmente, un file con estensione .xps. Rinominando il file in .zip è possibile notare (nella stessa maniera di un documento Office 2007) come questi siano costituiti da semplici file xml e binari per le risorse. Infatti, nel pacchetto, oltre al documento xml per il testo, vengono inclusi eventuali file necessari alla visualizzazione dello stesso come, ad esempio, immagini o font. Windows Presentation Foundation mette a disposizione dello sviluppatore una serie di API per la creazione di documenti XPS e, la base di tutto è proprio l'oggetto *FixedDocument*. Infatti un documento XPS è strutturato in maniera gerarchia ossia abbiamo una sequenza di *FixedDocument* raccolti tramite un *FixedDocumentSequence* ed ogni documento contiene una o più *FixedPage* che contengono la definizione di testo e immagini contenute nella pagina a cui si riferisce. Naturalmente una *FixedPage* viene definita tramite markup.

FATTURIAMO IN FORMATO XPS

Come abbiamo già accennato nell'introduzione nei casi in cui il nostro documento debba essere fedele al layout definito, occorre utilizzare un FixedDocument che, quindi, è progettato per applicazioni che richiedono una precisa rappresentazione del documento a prescindere dal supporto di visualizzazione. Un caso pratico di documento con queste caratteristiche è una fattura commerciale che deve, quindi, rispettare una precisa struttura da noi definita e non deve rimodellarsi in base al dispositivo come per un FlowDocument.
Iniziamo con il creare, quindi, l'oggetto **RigaFattura** che conterrà alcune proprietà comuni delle fatture come codice, descrizione, etc e una collection *Fattura* di tipo, naturalmente, *RigaFattura*.

```
public class RigaFattura
{
   private string codiceProdotto;
   private string descrizioneProdotto;
   private decimal prezzoUnitario;
   private Int32 quantita;
   public RigaFattura(string CodiceProdotto, string
        DescrizioneProdotto, decimal PrezzoUnitario, int
                                            Quantita)
   {
```

```
        this.codiceProdotto = CodiceProdotto;
        this.descrizioneProdotto = DescrizioneProdotto;
        this.prezzoUnitario = PrezzoUnitario;
        this.quantita = Quantita;
    }

    public string CodiceProdotto
    {
        get { return codiceProdotto; }
        set { codiceProdotto = value; }
    }

    public string DescrizioneProdotto
    {
        get { return descrizioneProdotto; }
        set { descrizioneProdotto = value; }
    }

    public decimal PrezzoUnitario
    {
        get { return prezzoUnitario; }
        set { prezzoUnitario = value; }
    }

    public int Quantita
    {
        get { return quantita; }
        set { quantita = value; }
    }
}
```

Niente di particolare come vedete. A questo punto definiamo la collection di questi elementi. Per la definizione di quest'ultima utilizzeremo un *ObservableCollection*. *ObservableCollection* è una novità introdotta con Windows Presentation Foundation e, a differenza delle classiche collection *Generic*, implementa già alcune nuove funzionalità di WPF come ad esempio le *DependencyProperty*. Nel nostro caso, visto che l'esempio sarà molto semplice, non è davvero necessaria ma, conviene sempre definire una collection in questo modo per, eventualmente, poter utilizzare in pieno le potenzialità di WPF.

```
public class Fattura :
                    ObservableCollection<RigaFattura>
{
    public Fattura()
    {
        this.Add(new RigaFattura("AAAAA", "T-Shirt
                    Gialla", (decimal)19.90, 2));
        this.Add(new RigaFattura("BBBBB", "T-Shirt
                    Nera", (decimal)19.90, 5));
        this.Add(new RigaFattura("CCCCC", "T-Shirt
                    Verde", (decimal)19.90, 1));
        this.Add(new RigaFattura("DDDDD", "Jeans
                    Uomo", (decimal)49.90, 3));
        this.Add(new RigaFattura("EEEEE", "Jeans
                    Donna", (decimal)39.90, 5));
    }
}
```

A questo punto possiamo definire la Window che visualizzerà la fattura. Per fare questo, naturalmente, utilizzare XAML definendo due pulsanti per la creazione della fattura e per la successiva creazione del file XPS della stessa e il *DocumentViewer* per la visualizzazione di quest'ultima.

```
<Window x:Class="FixedSample.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml
                    /presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
                    ml"
    Title="FatturazioneXPS" Height="463" Width="688"
    >
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="100"/> <!-- Riga
                    intestazione e pulsanti vari -->
      <RowDefinition/> <!-- Riga del DocumentViewer --
                    >
    </Grid.RowDefinitions>

    <!-- Intestazione e pulsanti vari -->
    <StackPanel Orientation="Vertical" Grid.Row="0">
      <Label Content="Fatturiamo con XPS"
          FontSize="30" Height="50" FontWeight="Bold"
```

COME INIZIARE

Per lo sviluppo del nostro progetto è necessario installare i seguenti elementi:

- **Visual Studio 2005: l'IDE dedicato alla programmazione in .NET**
- **Framework .NET 3.0: setup che installa le estensioni e gli assemblies di WPF, WCF e WWF**
- **Windows SDK: documentazione, libreria delle classi ed esempi su tutto ciò che riguarda la piattaforma Windows, tra il quale anche lo sviluppo con .NET 3.0**
- **Visual Studio 2005 extensions for .NET Framework 3.0 (WCF & WPF): estensioni del normale ambiente di sviluppo per supportare anche la programmazione di Windows Presentation Foundation.**

Tutto questo, ad eccezione di Visual Studio 2005, lo si può scaricare dal sito Microsoft.

In alternativa alla versione commerciale di VS, possiamo utilizzare la versione gratuita Express, scaricabile dal sito Microsoft.
E' necessario prestare attenzione all'ordine di installazione, che solitamente è quello indicato nella lista appena vista, ma che è bene leggere tra le note specifiche di ogni pacchetto di setup.
Per gli sviluppatori che utilizzano Visual Studio 2005 su Windows Vista, Microsoft ha sviluppato un aggiornamento di Service Pack 1, chiamato "Visual Studio 2005 Service Pack 1 Update for Windows Vista". Questo update rappresenta un ulteriore passo rispetto agli avanzamenti fatti con SP1 e mette a disposizione un'esperienza di alto livello per tutti gli sviluppatori che intendono sfruttare i vantaggi delle nuove caratteristiche di Windows Vista.

```
                HorizontalContentAlignment="Center"/>
      <StackPanel Orientation="Horizontal" Margin="0 5
                                                   0 5">
        <Button Content="Crea Fattura" Width="100"
                        Height="30" Click="CreaFattura"/>
        <Button Content="Crea XPS" Width="100"
                           Height="30" Click="CreaXPS"/>
      </StackPanel>
    </StackPanel>

    <!-- DocumentViewer per la visualizzazione della
                                            fattura -->
    <DocumentViewer x:Name="documentViewer"
              Background="AliceBlue" Grid.Row="1"/>
  </Grid>
</Window>
```

Nel code-behind andiamo a definire i vari metodi che utilizzeremo per creare e salvare la fattura in formato XPS. Come abbiamo già accennato in precedenza, un documento XPS è formato da un FixedDocument che a sua volta contiene una serie di *FixedPage*.
Assicuriamoci, innanzitutto, di importare i seguenti namespaces:

```
using System;
using System.IO;
using System.Printing;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Shapes;
using System.Windows.Xps;
using System.Windows.Xps.Packaging;
```

Con il seguente codice definiamo una *FixedPage* che sarà, quindi, la nostra fattura.

```
static FixedPage CreateFixedPage()
{
    FixedPage page = new FixedPage();
    page.Background = Brushes.White;
    page.Width = 816;
    page.Height = 1056;

    // Dati intestazione
    TextBlock tbIntestazione = new TextBlock();
    tbIntestazione.Text = "P I P P O s.r.l.";
    tbIntestazione.FontSize = 24;
    tbIntestazione.FontFamily = new
                              FontFamily("Verdana");
    FixedPage.SetLeft(tbIntestazione, 72);
    FixedPage.SetTop(tbIntestazione, 72);
    page.Children.Add(tbIntestazione);

    TextBlock tbIndirizzo=new TextBlock();
    tbIndirizzo.Text = "Corso Italia, 123";
    tbIndirizzo.FontSize = 20;
    tbIndirizzo.FontFamily=new FontFamily("Verdana");
    FixedPage.SetLeft(tbIndirizzo, 72);
    FixedPage.SetTop(tbIndirizzo, 95);
    page.Children.Add(tbIndirizzo);

    TextBlock tbCitta = new TextBlock();
    tbCitta.Text = "12345 Redmond (BA)";
    tbCitta.FontSize = 20;
    tbCitta.FontFamily = new FontFamily("Verdana");
    FixedPage.SetLeft(tbCitta, 72);
    FixedPage.SetTop(tbCitta, 117);
    page.Children.Add(tbCitta);

    // Data e Numero fattura
    TextBlock tbNumeroData = new TextBlock();
    tbNumeroData.Text = "Fattura n.2332 del
                                        31/08/2007";
    tbNumeroData.FontSize = 18;
    tbNumeroData.FontFamily = new
                              FontFamily("Verdana");
    FixedPage.SetRight(tbNumeroData, 72);
    FixedPage.SetTop(tbNumeroData, 72);
    page.Children.Add(tbNumeroData);

    // Dati del cliente
    TextBlock tbCliente = new TextBlock();
    tbCliente.Text = "Cliente: Vito Arconzo";
    tbCliente.FontSize = 18;
    tbCliente.FontFamily = new FontFamily("Verdana");
    FixedPage.SetRight(tbCliente, 72);
    FixedPage.SetTop(tbCliente, 95);
    page.Children.Add(tbCliente);

    // Linea orizzontale
  [...]
    // Intestazione dettaglio
    TextBlock tbCodice = new TextBlock();
    tbCodice.Text = "CODICE";
    tbCodice.FontSize = 16;
    tbCodice.FontFamily=new FontFamily("Tahoma");
    tbCodice.FontWeight = FontWeights.Bold;
    FixedPage.SetLeft(tbCodice, 72);
    FixedPage.SetTop(tbCodice, 180);
    page.Children.Add(tbCodice);

    TextBlock tbDescrizione = new TextBlock();
    tbDescrizione.Text = "DESCRIZIONE";
    tbDescrizione.FontSize = 16;
    tbDescrizione.FontFamily = new
                               FontFamily("Tahoma");
    tbDescrizione.FontWeight = FontWeights.Bold;
    FixedPage.SetLeft(tbDescrizione, 180);
    FixedPage.SetTop(tbDescrizione, 180);
    page.Children.Add(tbDescrizione);
```

RIFERIMENTI

...XML Paper Specification (XPS) è un formato per documenti. La caratteristica peculiare di esso è che può essere letto in qualsiasi hardware e da qualsiasi software. I documenti XPS possono essere stampati più velocemente, condivisi e archiviati anche in modo più sicuro. In Windows Vista è già presente il supporto per la generazione di documenti XPS mentre per XP e Windows Server è necessario installare il .NET Framework 3.0 o l'apposito viewer. Maggiori sul nuovo formato le trovate all'indirizzo www.microsoft.com/whdc/xps/default.mspx mentre il viewer per XP e Windows Server lo trovate all'indirizzo www.microsoft.com/whdc/xps/viewxps.mspx.

```
TextBlock tbQuantita = new TextBlock();
tbQuantita.Text = "QUANTITA'";
tbQuantita.FontSize = 16;
tbQuantita.FontFamily = new FontFamily("Tahoma");
tbQuantita.FontWeight = FontWeights.Bold;
FixedPage.SetLeft(tbQuantita, 430);
FixedPage.SetTop(tbQuantita, 180);
page.Children.Add(tbQuantita);

TextBlock tbPrezzoUnitario = new TextBlock();
tbPrezzoUnitario.Text = "PREZZO";
tbPrezzoUnitario.FontSize = 16;
tbPrezzoUnitario.FontFamily = new
                                FontFamily("Tahoma");
tbPrezzoUnitario.FontWeight = FontWeights.Bold;
FixedPage.SetLeft(tbPrezzoUnitario, 630);
FixedPage.SetTop(tbPrezzoUnitario, 180);
page.Children.Add(tbPrezzoUnitario);

// Linea orizzontale
                                                [...]

double top = 230;

Fattura fattura = new Fattura();
foreach (RigaFattura rigaFattura in fattura)
{
    TextBlock tbCodiceDetail = new TextBlock();
    tbCodiceDetail.Text = rigaFattura.CodiceProdotto;
    tbCodiceDetail.FontSize = 16;
    tbCodiceDetail.FontFamily = new
                                FontFamily("Tahoma");
    FixedPage.SetLeft(tbCodiceDetail, 72);
    FixedPage.SetTop(tbCodiceDetail, top);
    page.Children.Add(tbCodiceDetail);

    TextBlock tbDescrizioneDetail = new TextBlock();
    tbDescrizioneDetail.Text =
                        rigaFattura.DescrizioneProdotto;
    tbDescrizioneDetail.FontSize = 16;
    tbDescrizioneDetail.FontFamily = new
                                FontFamily("Tahoma");
    FixedPage.SetLeft(tbDescrizioneDetail, 180);
    FixedPage.SetTop(tbDescrizioneDetail, top);
    page.Children.Add(tbDescrizioneDetail);

    TextBlock tbQuantitaDetail = new TextBlock();
    tbQuantitaDetail.Text =
                        rigaFattura.Quantita.ToString();
    tbQuantitaDetail.FontSize = 16;
    tbQuantitaDetail.FontFamily = new
                                FontFamily("Tahoma");
    FixedPage.SetLeft(tbQuantitaDetail, 430);
    FixedPage.SetTop(tbQuantitaDetail, top);
    page.Children.Add(tbQuantitaDetail);

    TextBlock tbPrezzoUnitarioDetail = new
                                        TextBlock();
```

```
    tbPrezzoUnitarioDetail.Text =
                rigaFattura.PrezzoUnitario.ToString();
    tbPrezzoUnitarioDetail.FontSize = 16;
    tbPrezzoUnitarioDetail.FontFamily = new
                                FontFamily("Tahoma");
    FixedPage.SetLeft(tbPrezzoUnitarioDetail, 630);
    FixedPage.SetTop(tbPrezzoUnitarioDetail, top);
    page.Children.Add(tbPrezzoUnitarioDetail);

    top += 30;
  }

  return page;
}
```

Per la definizione del testo nella pagina utilizziamo semplicemente delle *TextBlock* e un oggetto *Line* per le linee di separazione della testata del dettaglio. Inoltre, per stampare il dettaglio utilizziamo un ciclo foreach sulla collection Fattura creata in precedenza. Nel successivo codice, invece, viene creato il FixedDocument che conterrà la *FixedPage* appena creata.

```
static FixedDocument CreateFixedDocument()
{
  FixedDocument doc = new FixedDocument();
  doc.DocumentPaginator.PageSize = new Size(96 *
                                        8.5, 96 * 11);

  PageContent page = new PageContent();
  FixedPage fixedPage = CreateFixedPage();
  ((IAddChild)page).AddChild(fixedPage);

  doc.Pages.Add(page);

  return doc;
}
```

Al FixedDocument creato, quindi, viene aggiunta la *FixedPage* con i dati della fattura attraverso il metodo *Add()* della collection *Pages* del documento. Fatto questo possiamo definire gli eventi dei pulsanti che andranno a richiamare i metodi appena definiti. Gestiamo l'evento *Click()* del pulsante che crea il documento e lo visualizza nel *DocumentViewer*.

```
void CreaFattura(object sender, EventArgs e)
{
  // Creazione del FixedDocument ...
  doc = CreateFixedDocument();
  documentViewer.Document = doc;
}
```

A questo punto, cliccando sul pulsante "Crea Fattura", verrà creato il documento come mostrato in figura 2.

NOTA

TIPI DI DOCUMENTO

In Windows Presentation Foundation abbiamo a disposizione due tipologie di documenti: FlowDocument e FixedDocument. I primi hanno la particolarità di adattare il contenuto automaticamente in base al device su cui vengono visualizzati riadattando le colonne o i paragrafi. I FixedDocument, invece, sono documenti con struttura fissa e utili, quindi, a situazioni in cui il layout non deve modificarsi a prescindere dal device. Nel nostro esempio abbiamo utilizzato un fattura che, naturalmente, deve avere un layout ben definito.

L'AUTORE

Vito Arconzo, sviluppa applicazioni sia Windows Forms che Web basate sul Microsoft .NET Framework già dalle prime versioni beta. In questo momento ciò che lo appassiona maggiormente è il nuovo engine introdotto con la versione 3.0 del .NET Framework, Windows Presentation Foundation ed è, inoltre, membro dello User Group DotNetSide (http://www.dotnetside.org). E' possibile contattarlo attraverso il suo blog all'indirizzo http://blog.vitoarconzo.it per qualsiasi dubbio riguardante l'articolo o WPF in generale.

La nostra fattura è finalmente creata, in maniera abbastanza semplice e già con le funzionalità di base del DocumentViewer possiamo operare su di essa con, ad esempio, zoom o navigazione tra pagine. La cosa interessante è che effettuando lo zoom su di essa, il testo viene nuovamente ridisegnato per far si che la qualità non venga persa. Infatti, come tutto in Windows Presentation Foundation, anche i documenti sono vettoriali e, soprattutto, le dimensioni sono specificate in *dip* (Device Independent Pixel) che permetto una maggiore resa a qualsiasi risoluzione.
Il prossimo passo è quello di salvare il documento su disco in formato XPS, operazione altrettanto semplice. Per fare questo occorre, naturalmente, creare un nuovo oggetto *XpsDocument* indicando dove il file verrà creato e, attraverso l'utilizzo di un oggetto *XpsDocumentWriter* scrivere il documento su disco. Naturalmente questo viene fatto nell'evento Click() del pulsante "Crea XPS".

```
void CreaXPS(object sender, EventArgs e)
{
    if(doc == null){return;}
    try
    {
        XpsDocument xpsd = new
  XpsDocument(@"c:\prova.xps", FileAccess.ReadWrite);
        XpsDocumentWriter xw =
            XpsDocument.CreateXpsDocumentWriter(xpsd);
        xw.Write(doc);
        xpsd.Close();
        MessageBox.Show("XPS Creato!");
    } catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Nel nostro esempio, il file creato viene chiamato prova.xps e si trova in c:\. Cliccando sul file viene automaticamente aperta una nuova istanza di Internet Explorer che visualizza il documento come in figura 3.
Un altro aspetto importante fornito da Windows Presentation Foundation è la possibilità di memorizzare all'interno di un documento uno o più *PrintTicket*. Si tratta di una serie di proprietà che indicano come un'intera sequenza di documenti, un singolo documento o una singola pagina devono essere stampati. Possono essere specifiche ad un particolare device, in funzione delle *PrintCapabilities*, oppure generiche e hanno un loro scope d'azione, a livelli, partendo dalla *FixedDocumentSequence* fino alla *FixedPage*. Possiamo, ad esempio, specificare che l'intera stampa va effettuata a qualità testuale, mentre una specifica *FixedPage*, poiché contenente immagini, vada stampata a qualità fotografica. Oppure alcune pagine possono essere specifiche per essere stampate in orizzontale piuttosto che in verticale. Oltre a questo possiamo controllare la risoluzione, i margini, il numero di copie e la dimensione della stampa.
Per sfruttare questa caratteristica dobbiamo creare un oggetto *PrintTicket* e passarlo al metodo Write dell'oggetto *XpsDocumentWriter*.

```
PrintTicket ticket = new PrintTicket();
ticket.OutputQuality = OutputQuality.Text;
ticket.PageOrientation = PageOrientation.Portrait;
xw.Write(doc.DocumentPaginator, ticket);
```

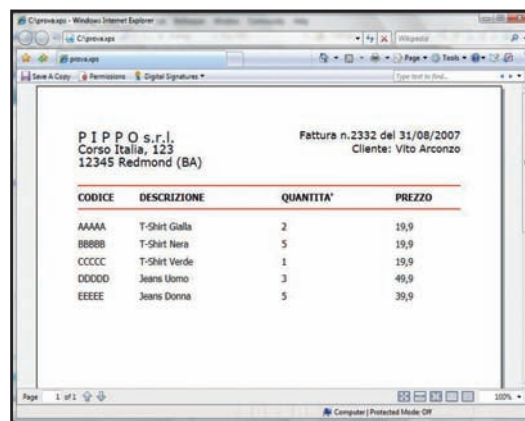



Figura 3: La nostra fattura visualizzata in Internet Explorer

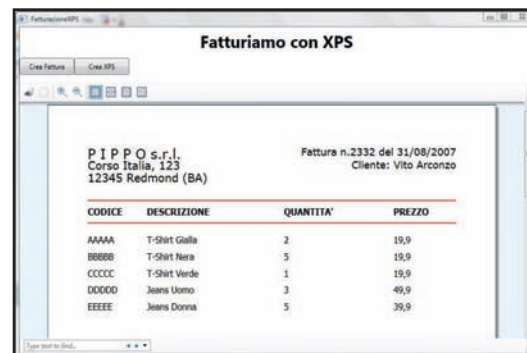



Figura 2: La nostra fattura visualizzata nel DocumentViewer

CONCLUSIONI

. Il formato XPS, stesso, nasce come alternativa all'ormai diffusissimo PDF ed in questo senso sembra essere partito con il piede giusto. È quindi utile cominciare a conoscerlo e sfruttarlo anche per le proprie applicazioni. In ogni caso sicuramente ne sentiremo parlare ancora, vedremo se scalfirà lo strapotere del PDF

Vito Arconzo

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

PHP

COME POSSO CONOSCERE LA DIFFERENZA FRA DUE DATE PASSATE COME STRINGHE?

```
function date_diff_as_text($ts1, $ts2) {

 /*
 $ts1 = "2007-01-05 10:30:45";
 $ts2 = "2007-01-06 10:31:46";
 echo date_diff_as_text($ts1, $ts2);
 */

 $ts1 = strtotime($ts1);
 $ts2 = strtotime($ts2);
 $diff = abs($ts1-$ts2);

 $sec_min = 60;
 $sec_hour = $sec_min*60;
 $sec_dias = $sec_hour*24;

 $dias = intval($diff/$sec_dias);
 $hours = intval($diff/$sec_hour)%24;
 $minutes = intval($diff/$sec_min)%60;
 $seconds = $diff%60;

 if ($dias > 0) {
   $result = "$days day";
   if ($dias > 1) {
     $result .= "s";
   }
 }

 if ($hours > 0) {
   $result .= " $hours hour";
   if ($hours > 1) {
     $result .= "s";
   }
 }

 if ($minutes > 0) {
   $result .= " $minutes minute";
   if ($minutes > 1) {
     $result .= "s";
   }
 }

 if ($seconds > 0) {
   $result .= " $seconds second";
   if ($seconds > 1) {
     $result .= "s";
   }
 }

 $result = explode(" ", $result);
 if (count($result)>2) {
   end($result);
   $key1 = key($result);
   prev($result);
   $key2 = key($result);
   $aux = $result[$key2];
   $aux .= " ".$result[$key1];
   unset($result[$key1]);
   unset($result[$key2]);
   $result = implode(" ", $result);
   $result .= " y $aux";
 } else {
   $result = implode(" ", $result);
 }

 return $result;
}
```

COME POSSO CONOSCERE L'INTERVALLO DI TEMPO TRASCORSO FRA DUE TIMESTAMP?

```
function callDuration($dateTimeBegin,$dateTimeEnd) {

    $dif=$dateTimeEnd - $dateTimeBegin;

    $hours = floor($dif / 3600);
    $temp_remainder = $dif - ($hours * 3600);
```

```
    $minutes = floor($temp_remainder / 60);
    $temp_remainder = $temp_remainder - ($minutes * 60);

    $seconds = $temp_remainder;

    // leading zero's - not bothered about hours
    $min_lead=':';
   if($minutes <=9)
      $min_lead .= '0';
    $sec_lead=':';
   if($seconds <=9)
      $sec_lead .= '0';

 // difference/duration returned as Hours:Mins:Secs e.g. 01:29:32

 return $hours.$min_lead.$minutes.$sec_lead.$seconds;

     }
```

C#

COME POSSO CONOSCERE LA LISTA DI TUTTI I DATABASE CONTENUTI IN UN'ISTANZA DI SQL SERVER?

```
function callDuration($dateTimeBegin,$dateTimeEnd) {

    $dif=$dateTimeEnd - $dateTimeBegin;

    $hours = floor($dif / 3600);
    $temp_remainder = $dif - ($hours * 3600);

    $minutes = floor($temp_remainder / 60);
    $temp_remainder = $temp_remainder - ($minutes * 60);

    $seconds = $temp_remainder;

    // leading zero's - not bothered about hours
    $min_lead=':';
   if($minutes <=9)
      $min_lead .= '0';
    $sec_lead=':';
   if($seconds <=9)
      $sec_lead .= '0';
 // difference/duration returned as Hours:Mins:Secs e.g. 01:29:32
 return $hours.$min_lead.$minutes.$sec_lead.$seconds;
     }
```

COME POSSO CONVERTIRE UNA BMP NELLA SUA RAPPRESENTAZIONE ASCII?

```
StreamWriter sw = File.CreateText(outputPath);
Color color;
//Start of html file
sw.Write("<html> \n <body > \n <pre style=\"font: 10px/3px
                                              monospace;\">");
Random ran = new Random();
for (int y = 0; y < bmp.Height; y=y+3)
{
    for (int x = 0; x < bmp.Width; x=x+3)
    {
        int random = ran.Next(2);
        color = bmp.GetPixel(x, y);
        //write the html tag corresponding to the pixel color
        //value of random will be either 1 or 0
        sw.Write("<span style=\"color:
  #{0:X2}{1:X2}{2:x2};\">{3}</span>", (int)color.R, (int)color.G,
                                            (int)color.B, random);
    }
    sw.WriteLine("<br>");
}

//End of html file
sw.Write("</body>\n</html>");

sw.Close();

    End Sub
```

COME POSSO CONVERTIRE DA HTML AD RTF

```
Private Sub ConvertHtmlToRTF()
' I sample to read from html file
Dim sHtml As String = IO.File.ReadAllText("C:\TestHtml.htm")

' I will save it to temp folder
If Dir("C:\Temp", FileAttribute.Directory) = "" Then
System.IO.Directory.CreateDirectory("C:\Temp")
End If

' Create rtf File
Dim b As System.IO.TextWriter = New
                System.IO.StreamWriter("C:\Temp\test.rtf", False,
                                  System.Text.Encoding.Unicode)
' Write it to rtf format
b.Write(sHtml)

' Clear object
b.Flush()
b.Close()
' Open file
System.Diagnostics.Process.Start("C:\Temp\test.rtf")

End Sub
```

POSIZIONAMENTO DEI CONTROLLI

COME CREARE APPLICAZIONI CHE SCELGONO LA DISPOSIZIONE DEI COMPONENTI SULLA BASE DELLA RISOLUZIONE DEL SISTEMA O DELLA FINESTRA GRAFICA? ECCO LE TECNICHE PER NON DOVER MAI RICORRERE A FORM NON RIDIMENSIONABILI

Benvenuti al nostro terzo appuntamento con wxWidgets. Nello scorso numero abbiamo fatto grandi passi avanti, e ora siamo in grado di creare finestre, riempirle di controlli a nostro piacimento, e far rispondere l'applicazione alle richieste dell'utente per mezzo delle *Event Table*. Se non ci credete, date un'occhiata al *wxTelefono* in figura 1: l'abbiamo fatto tutto da soli la volta scorsa. A dir la verità ci sono dei particolari di quest'esempio che ancora non vanno molto bene, e in questa puntata vedremo di migliorarlo. Allargheremo così le nostre vedute a nuovi modi di intendere la disposizione dei controlli, impareremo a creare interfacce complesse col minimo sforzo, e introdurremo molte altre tecniche e caratteristiche di wxWidgets.

Fig. 1: L'applicazione wxTelefono, introdotta nello scorso appuntamento

IL VECCHIO WXTELEFONO

Partiamo subito chiarendo cosa non mi piace del nostro vecchio *wxTelefono*. Semplificherò la vita ai pigri e a chi non ha seguito gli scorsi appuntamenti (male!), riprendendo qui uno stralcio significativo del codice. Si tratta del costruttore del frame, in cui i controlli (cioè il display e i pulsanti) vengono creati e disposti al suo interno.

```
PhoneFrame::PhoneFrame() :
  wxFrame(0, wxID_ANY, _T("wxTelefono"),
  wxPoint(0,0), wxSize(188, 245)) {
    //--- Crea il display ---
    const wxSize DisplaySize(180, 50);
    display_ = new wxTextCtrl (
        this, wxID_ANY, _T(""),
        wxPoint(0,0), DisplaySize
    );
    //Lo rende impossibile da modificare
    display_->SetEditable(false);
    //--- Crea i pulsanti ---
    const wxSize ButtonSize(60, 40);

    const wxString label[4][3] = {
        {_T("1"), _T("2"), _T("3")},
        {_T("4"), _T("5"), _T("6")},
        {_T("7"), _T("8"), _T("9")},
        {_T("*"), _T("0"), _T("#")}
    };

    for (unsigned int y=0; y<4; ++y) {
        for (unsigned int x=0; x<3; ++x) {
            new wxButton (
                this, ID_BUTTON, label[y][x],
                wxPoint (
                    x * ButtonSize.x,
                    DisplaySize.y + y * ButtonSize.y
                ),
                ButtonSize
            );
        }
    }
}
```

Il codice crea un frame di dimensioni 188x245, con un display di dimensioni 180x50, e i due cicli for dispon-

Impegno

Tempo di realizzazione

gono una serie di pulsanti (60x40 ciascuno) in una griglia prefissata.

I PROBLEMI DEL POSIZIONAMENTO ASSOLUTO

Questo tipo di gestione dei controlli viene definito posizionamento assoluto, e viene adottato da molti sistemi RAD, fra i quali è emblematico l'esempio di Visual Basic. Il posizionamento assoluto, infatti, è intuitivo e facile da implementare.
Purtroppo la scarsa fama di cui godono i programmatori Visual Basic per quanto riguarda la flessibilità delle interfacce, testimonia il fatto che il posizionamento assoluto impone alle applicazioni diversi svantaggi:

- **Scarsa portabilità**: non esiste alcuno standard che regoli le forme e le dimensioni dei controlli. Ogni sistema operativo usa pulsanti e finestre diversi, con bordi più o meno grandi, diversa gestione dei menù, delle barre, eccetera. Pertanto se usate il posizionamento assoluto, preparatevi a dozzine di e-mail di utenti insoddisfatti perché nel loro irrinunciabile window manager il vostro programma diventa inutilizzabile.
- **Dipendenza dalla risoluzione**: anche all'interno dello stesso sistema operativo, risoluzioni video differenti da quella in cui il sistema è stato progettato possono mandare all'aria le dimensioni precalcolate, costringendovi a ridicoli messaggi del tipo: "*per quest'applicazione è consigliata/necessaria una risoluzione di 800x600*".
- **Scarsa flessibilità**: gli utenti vogliono personalizzare la loro applicazione, organizzando il contenuto e la disposizione dei controlli. Il posizionamento assoluto non glielo permette, restituendogli invece un mondo preconfigurato e immutabile.
- **Codice illeggibile e ferraginoso**: il posizionamento assoluto impone la memorizzazione di costanti magiche all'interno del codice sorgente, peggiorandone la leggibilità. Il codice diventa inoltre difficile da manutenere: piccole variazioni del design dell'applicazione ("*Aggiungiamo un secondo display un po' più in basso*") portano inevitabilmente ad una cascata di modifiche ad-hoc ("*Allora dobbiamo cambiare il posizionamento di tutti i pulsanti, allungare la finestra, e…*").

Per queste (e altre!) ragioni, **quella del posizionamento assoluto è un'idea ingenua e dannosa**. Questo è vero soprattutto nel nostro caso, dato che se usate wxWidgets probabilmente desiderate che il vostro programma sia cross-platform. Pertanto: rinunciate anche solo all'idea di poter comunicare al programma delle dimensioni assolute.

Le vie più sensate, invece, sono fondamentalmente due, spesso in sinergia fra loro:

1) **Affidare il layout al programma**, dandogli eventualmente dei suggerimenti su come suddividere lo spazio fra i vari controlli in termini di proporzioni, e sulle dimensioni minime e massime.

2) **Affidare il layout all'utente**. In questo modo non solo non dovrete preoccuparvi di studiarne uno perfetto, ma farete anche felici i destinatari della vostra applicazione, permettendo loro di personalizzarla secondo le proprie necessità e il grado esperienza che hanno raggiunto.

In questa puntata vedremo alcuni degli strumenti che wxWidgets offre per raggiungere questi encomiabili obiettivi: **sizer**, **contenitori** per il posizionamento dinamico (come **wxPanel** e **wxSplitterWindow**), e veri e propri framework per il layout avanzato dei componenti (come **wxAui**).

UTILIZZIAMO I SIZER

Il concetto alla base dei sizer è un po' un classico dei pattern ad oggetti, e quindi suonerà familiare ad alcuni programmatori che vengono da altri framework – ad esempio ai Javisti che vengono da *AWT*.
L'idea è di affidare i controlli a degli oggetti che si occupino di organizzarne la disposizione all'interno della finestra automaticamente. In wxWidgets tali oggetti intelligenti vengono chiamati sizer e derivano tutti dalla classe base **wxSizer**. Il sizer più semplice è **wxBoxSizer**, che si occupa di disporre i controlli che gli vengono affidati in fila (se orientato orizzontalmente) o in colonna (se orientato verticalmente).
Per vedere un sizer in azione, cominciamo a ritoccare il costruttore di *PhoneFrame*:

```
PhoneFrame::PhoneFrame() :
  wxFrame(0, wxID_ANY, _T("wxTelefono"))
{
    //creiamo un BoxSizer verticale
    wxBoxSizer* sizer = new
                                wxBoxSizer(wxVERTICAL);
  //lo impostiamo come sizer principale del frame
 SetSizer(sizer);

 //Creiamo il display
 display_ = new wxTextCtrl (this, wxID_ANY, _T(""));
  //Lo aggiungiamo al sizer
 s->Add(display_, 1);
}
```

Abbiamo creato un *wxBoxSizer* verticale, che abbiamo associato al frame come sizer principale, grazie alla chiamata a *SetSizer*. Questo permetterà alla fine-

NOTA

FLICKERING!
Anche se qui non è stato fatto per ragioni di spazio, quando si usa wxAui, è buona norma definire gli eventi OnSize e OnEraseBackground del frame ospite con un corpo vuoto, per evitare fastidiosi sfarfallii in caso di ridimensionamento. L'esempio su CD riporta il codice completo.

stra di ridimensionare il sizer (e quindi tutti i controlli ad esso affidati) ogniqualvolta sarà necessario.
Quindi affidiamo il display al sizer, richiamando la funzione membro Add. Il risultato dell'esecuzione è visibile in figura 2a.

Fig. 2: Il display
a) Con un wxBoxSizer verticale
b) Più WX_EXPAND
c) Più un bordo applicato a tutti e quattro i lati

per modificare le impostazioni di allineamento del controllo.
Notate, infine, che il display si espande fino a toccare il bordo della finestra. Se vogliamo associare uno o più margini, possiamo usare il quarto parametro.

```
s->Add(display_, 1, wxEXPAND | wxALL, 20);
```

Il risultato è visibile in figura 2c.

Notate che abbiamo dovuto aggiungere anche un flag, *wxALL*, che indica di applicare il bordo a tutti i lati. In effetti la chiamata non è che un'abbreviazione di:

```
s->Add(display_, 1, wxEXPAND | wxUP |
                    wxRIGHT | wxDOWN | wxLEFT, 20);
```

Se si vuole applicare il bordo solo ad alcuni dei lati, quindi, è possibile indicarli per composizione.

I PARAMETRI DI WXSIZER::ADD

La funzione membro *wxSizer::Add* è importantissima e molto flessibile, grazie al gran numero di parametri e di cui dispone. La marcatura (o meglio, la parte più importante della marcatura!) è questa:

```
wxSizerItem* Add(wxWindow* controllo, int
          proporzione = 0, int flag = 0, int bordo = 0)
```

Vediamo un po' in che modo i vari parametri influenzano il comportamento del sizer.
Notate che nella figura 2 il sizer ha allungato automaticamente il display, in modo da fargli riempire verticalmente tutto lo spazio della finestra. Se provate a ridimensionare il frame, il display lo seguirà accorciandosi. Questo comportamento è dovuto al fatto che abbiamo passato 1 come secondo parametro. Vedremo meglio il comportamento di ***proporzione*** più avanti: per ora vi basti sapere che se non volete che il controllo si espanda, basterà passare 0 come secondo parametro, oppure non passarlo affatto.
Notate anche che il display si espande ***solo*** verticalmente, ma non orizzontalmente. Ciò deriva dal fatto che il sizer organizza lo spazio secondo il suo orientamento – e in questo caso è stato costruito come wxVertical. Se volete far crescere il controllo anche nell'altra direzione, potete impostare il terzo parametro su ***wxEXPAND***. Se nel codice precedente, quindi, modifichiamo la chiamata ad Add così:

```
s->Add(display_, 1, wxEXPAND);
```

il risultato sarà quello in figura 2b.

wxEXPAND è uno dei tanti **flag** che si possono associare,

GESTIRE LE PROPORZIONI

Giunti fin qui, direi che è ora di smetterla di strapazzare quel povero display! Dobbiamo costruire con i sizer tutto il wxTelefono. Concentriamoci sul piano di lavoro, visibile in Figura 3.

Fig. 3: Organizzazione degli spazi nel sizer

La figura mostra la suddivisione degli spazi all'interno della finestra: notate che è stata disegnata su carta quadrettata, per farvi calcolare meglio le proporzioni – a noi interessano solo quelle verticali.

- 3 quadretti per il display.
- 2 quadretti per uno spazio intermedio
- 14 quadretti per la pulsantiera

Ora potete capire meglio lo scopo del secondo parametro di wxSizer::Add: indicare le proporzioni. Per il display, ad esempio, scriveremo:

```
s->Add(display_, 3, wxEXPAND);
```

BIBLIOSITOGRAFIA

Fate riferimento alla documentazione ufficiale (http://wxwidgets.org/docs/) per avere informazioni su classi e funzioni della libreria.

Se, come me, preferite i libri di carta, Cross-Platform Gui Programming with wxWidgets, scritto da Julian Smart (lo stesso autore della libreria), è un buon punto di partenza e un primo punto di riferimento per il framework.

Infine, ricordate che wxWidgets è open source, e spesso il codice sorgente è la migliore delle documentazioni!

Indicando per gli altri elementi le proporzioni 2 e 14, il sizer sarà in grado di calcolare e distribuire lo spazio fra i componenti. Questo ci consente di lasciare inalterato l'aspetto grafico della nostra applicazione in caso di resize.

SPAZIATORI

Il secondo elemento della lista è uno spazio vuoto, di due quadretti. Potremmo crearlo facilmente usando un controllo trasparente, come uno StaticText con label vuota.

In realtà wxWidgets permette di cavarsela molto più agevolmente, senza dover creare alcun inutile controllo: basta aggiungere uno **spaziatore**, richiamando le funzioni membro ***AddSpacer*** e ***AddStretchSpacer***. Quest'ultimo crea uno spaziatore che si espande secondo la proporzione indicata nel parametro. Nel nostro caso, useremo:

```
s->AddStretchSpacer(2);
```

SIZER NIDIFICATI

Ora rimane solo l'ultimo elemento, la pulsantiera. In effetti, è anche il più complicato: sono dodici pulsanti, disposti su quattro file. Come risolvere il problema? Il trucco è: **annidare altri sizer**. Un sizer, infatti, può tranquillamente contenerne un altro di qualsiasi tipo – è una tecnica che viene usata spesso.

Annidando più sizer potremmo cavarcela piuttosto agevolmente anche soltanto con gli strumenti visti finora. Ad esempio, potremmo seguire la figura 4: usare un sizer verticale, contenente quattro sizer orizzontali, contenenti tre pulsanti ciascuno. Se volete impratichirvi con wxWidgets, vi invito calorosamente ad implementare questa soluzione, ma noi qui ne seguiremo una ancora più semplice: useremo un **wxGridSizer**. Non dimenticatevi, infatti, che esistono eredi

Fig. 4: Come costruire la pulsantiera con una gerarchia di wxBoxSizer innestati.

di wxSizer per tutte le esigenze! Un *wxGridSizer* si occupa automaticamente di gestire una griglia di n x m controlli. Le dimensioni vengono passate all'interno del costruttore, e i controlli vengono incasellati uno alla volta, dall'alto in basso e da sinistra a destra, ogni volta che si richiama la funzione Add.

WXTELEFONO CON I SIZER

Ecco, quindi, il codice del costruttore del nostro nuovo wxTelefono, ottenuto grazie ai sizer:

```
PhoneFrame::PhoneFrame() :
  wxFrame(0, wxID_ANY, _T("wxTelefono")) {
    wxPanel* panel = new wxPanel(this);
    wxBoxSizer* sizer = new
wxBoxSizer(wxVERTICAL);

    panel->SetSizer(sizer);
    //Aggiunge il display
    display_ = new wxTextCtrl (panel, wxID_ANY,
_T(""));
    display_->SetEditable(false);
    sizer->Add(display_, 3, wxEXPAND);

    //Aggiunge lo spaziatore
    sizer->AddStretchSpacer(2);
    //Aggiunge la pulsantiera
    wxGridSizer* buttonSizer = new wxGridSizer(3,
4);
    sizer->Add(buttonSizer, 14, wxEXPAND);
    const wxString labels(_T("123456789*0#"));

    for (unsigned int i=0; i < labels.length(); ++i) {
          wxButton* button = new wxButton(
              panel, ID_BUTTON, wxString(_T("&")) +
labels[i]
          );
         buttonSizer->Add(button, 1, wxEXPAND |
wxALL, 3);
    }
}
```

Dal codice sono sparite tutte quelle orrende costanti magiche, e il ciclo for si è notevolmente ridotto. Grazie a *wxGridSizer*, infatti, i controlli vengono inseriti in sequenza e non è più neanche necessario memorizzare le label in una matrice: una semplice stringa è sufficiente. Il risultato potete vederlo in Figura 5. Sembra pure più bello, non vi pare?

USIAMO UN WXPANEL

D'accordo, il telefono in Figura 5 è effettivamente più bello, e ci sono cose del codice che non ho spie-

Fig. 5: wxTelefono implementato con sizer e pannelli.

gato. Se non vi siete accorti di niente (male!), vi faccio notare che lo sfondo della finestra è cambiato in un colore più salutare, che sotto ai numeri sono comparse delle strane barrette, e che il merito è di quel controllo *wxPanel* che ho infilato nel codice senza che ve ne accorgeste. Quella di associare un pannello ad un *wxFrame*, dovrebbe essere considerata una pratica **obbligatoria**. La ragione sta nel fatto che alcuni window manager non supportano bene il disegno diretto su un *wxFrame* e potrebbero fare strani scherzi. Inoltre, *wxPanel* implementa anche tante funzionalità di base che potrebbero non trovarsi nella finestra ospite. Ad esempio, vi siete accorti che in un *wxFrame* non potete usare i tasti freccia e tab per spostarvi da un pulsante all'altro? Con un *wxPanel* invece, sì. Un *wxPanel*, infatti, reagisce automaticamente agli eventi della tastiera, e questo ha a che fare direttamente con quelle barrette sotto i numerini. Se guardate il codice all'interno del ciclo for, noterete l'istruzione:

```
wxString(_T("&")) + labels[i]
```

Con questa concatenazione, ogni pulsante si trova ad avere come label il proprio numero, preceduto dal simbolo di ampersand. Questo simbolo ha un significato molto specifico in wxWidgets: serve ad indicare una scorciatoia da tastiera, applicata al carattere immediatamente successivo. Così, potete scrivere un numero di telefono via tastiera, senza toccare il mouse. Il merito, dietro le scene, è proprio del pannello, che si occupa di gestire gli eventi. Come vedete, i pannelli sono molto più utili di quanto non possano sembrare a prima vista. Potete usarli ogni volta che avete dei gruppi di controlli ai quali volete dare un corpo unitario, trattandoli come se fossero una finestra a tutti gli effetti.

WXSPLITTERWINDOW

Grazie ai sizer, quindi, siamo stati in grado di realizzare la nostra prima strategia: affidare il layout dei controlli alla nostra stessa applicazione.
Come abbiamo visto, una strategia ancora più furba, versatile ed efficace è lasciare quest'incombenza all'utente finale. Perfino nel nostro esempio, per quanto minimo, qualche utente potrebbe non gradire le proporzioni che abbiamo predisposto. Alcuni preferiranno, magari, un telefono come quello mostrato in Figura 6, con un display molto lungo e una pulsantiera ridotta, in basso.
Per lasciare che sia l'utente a scegliere una fra le infinite proporzioni possibili, possiamo usare un contenitore speciale: **wxSplitterWindow**. Un *wxSplitterWindow* contiene esattamente due finestre (ovverosia due *wxWindow* o derivati), separate - orizzontalmente o verticalmente - da un piccolo bordo che l'utente può trascinare a piacimento, modificando così il rapporto fra le loro dimensioni. Sembra fatto apposta per usarlo nel nostro caso: separeremo il display dalla pulsantiera. C'è un problema, però: mentre il display è un controllo (ovverosia una classe derivata da *wxWindow*), la pulsantiera è un semplice wxGridSizer. Quindi non possiamo metterla direttamente nello splitter. Se avete letto attentamente il paragrafo precedente, avrete già intuito la soluzione: i wxPanel sono i migliori amici dei designer.

Fig. 6: L'utente ha allungato il display, grazie a wxSplitterWindow.

```
PhoneFrame()::PhoneFrame() :
  wxFrame(0, wxID_ANY, _T("wxTelefono")) {
    //crea la splitter window
    wxSplitterWindow* splitter = new
                          wxSplitterWindow(this);
    //Crea il display
    display_ = new wxTextCtrl (splitter,
                          wxID_ANY, _T(""));
      //Crea un pannello con la pulsantiera
    wxPanel* buttonPanel = new
```

```
                                    wxPanel(splitter);
    wxGridSizer* buttonSizer = new
                                    wxGridSizer(3, 4);
    buttonPanel->SetSizer(buttonSizer);

    for (unsigned int i=0; i < labels.length(); ++i) {
        wxButton* button = new wxButton(
            buttonPanel, ID_BUTTON,
                        wxString(_T("&")) + labels[i]
        );
          buttonSizer->Add(button, 0, wxEXPAND);
    }
    //inserisce display e pulsantiera nello splitter
    splitter->SplitHorizontally(display_,
                                    buttonPanel, 0);
}
```

Come vedete dal codice, l'unico cambiamento sostanziale è l'aggiunta dello splitter, e l'inserimento dei pulsanti in *buttonPanel*.
Infine, una volta creati i controlli, è necessario indicare allo splitter di fare il suo lavoro: richiamiamo la funzione membro *SplitHorizontally*, per creare una bella divisione modificabile fra display e buttonPanel.

MASSIMO RIUTILIZZO

Seguendo il principio esposto nel paragrafo precedente, è possibile costruire intere applicazioni dai controlli ridimensionabili. Un metodo potrebbe essere quello di utilizzare delle gerarchie di oggetti *wxSplitterWindow*, proprio come si fa normalmente con i sizer. Il risultato sarebbe senz'altro interessante e flessibile, ma probabilmente il codice e la progettazione risentirebbe un po' di tante suddivisioni. Rendere i controlli ridimensionabili, poi, non è certo sufficiente. Occorre anche che l'utente possa salvare in qualche maniera le sue preferenze, in modo da trovare tutto a posto ad ogni riavvio del programma e non essere costretto, ogni volta, ad un inutile lavoro manuale. E per dirla tutta, nei programmi più avanzati, il semplice ridimensionamento è il primo passo. I pannelli e i componenti possono essere spostati in giro per lo schermo, ancorati ai bordi della finestra, e incastrati l'uno all'altro in modo da formare blocchi di controlli ridimensionabili, in maniera assolutamente dinamica. Se programmate con Visual Studio Express, avete un ambiente simile proprio davanti a voi: potete spostare le numerose finestre dell'IDE, ancorarle, chiuderle, riaprirle… Creare un simile meccanismo a mano con wxWidgets, per ogni applicazione, sarebbe uno sforzo immane. Proprio per questo sono nati diversi progetti per permettere il riutilizzo di interfacce avanzate. Il più avanzato si chiama ***wxAui***, ed è stato inserito ufficialmente in wxWidgets soltanto di recente. Grazie a *wxAui* possiamo inserire il display e la tastiera di *wxTelefono* in un vero e proprio ambiente gestito.

UN WXTELEFONO "GESTITO"

Per poter programmare con wxAui è necessario includere il file "**wx/aui/aui.h**", e aggiungere all'elenco delle librerie in ingresso la voce: **wxmsw28d_aui.lib** (o la libreria corrispondente secondo la vostra configurazione. Ad esempio, per la versione debug unicode: **wxmsw28ud_aui.lib**). Il cardine di wxAui è la classe ***wxAuiManager***, che viene associata ad un frame principale (il nostro *PhoneFrame*) e che si occupa di "gestire" una serie di finestre o pannelli (i nostri *display_* e *buttonPanel*). La nostra classe wxPhoneFrame, quindi, incorporerà un nuovo oggetto (che chiameremo *auiManager*) di tipo *wxAuiManager*, e il suo costruttore cambierà così:

```
PhoneFrame::PhoneFrame() :
  wxFrame(0, wxID_ANY, _T("wxTelefono")) {
      auiManager.SetManagedWindow(this);
      //Aggiunge il display
      display_ = new wxTextCtrl(this, wxID_ANY,
                                          _T(""));
      display_->SetEditable(false);
      auiManager.AddPane(display_);
      auiManager.GetPane(display_).
           Name(_T("Display")).
           Caption(_T("Display")).
           PinButton(true);
      //Aggiunge la pulsantiera
      wxPanel* buttonPanel = new wxPanel(this);
      wxGridSizer* buttonSizer = new
                                 wxGridSizer(3, 4);
      buttonPanel->SetSizer(buttonSizer);
      const wxString labels(_T("123456789*0#"));
              for (unsigned int i=0; i <
                          labels.length(); ++i) {
          wxButton* button = new wxButton(
              buttonPanel, ID_BUTTON,
                        wxString(_T("&")) + labels[i]
          );
                     buttonSizer->Add(button, 0,
                                        wxEXPAND);
      }
      auiManager.AddPane(buttonPanel);
      auiManager.GetPane(buttonPanel).
          Name(_T("Buttons")).
          Caption(_T("Buttons"))..
          PinButton(true);
      auiManager.Update();
  }
```

Come vedete, si tratta di poche modifiche, ma di grande effetto. Innanzitutto abbiamo associato il manager al nostro PhoneFrame tramite una chiamata a *auiManager.SetManagedWindow*.
Poi, abbiamo creato dei *pane* contenenti il display e la pulsantiera, tramite le chiamate ad *auiManager.AddPane*. Questa funzione richiede come primo para-

metro la finestra da gestire, e come secondo parametro l'elenco delle proprietà del pane, sotto forma di *wxPaneInfo*. *wxPaneInfo*, infatti, è una classe che contiene una sfilza di funzioni membro (*Name, Caption, PinButton…*) ognuna delle quali permette di impostare una specifica caratteristica del *pane*. E ogni funzione membro restituisce a sua volta un oggetto di tipo *wxPaneInfo*, permettendo così una curiosa sintassi, grazie alla quale è possibile cambiare una fila di proprietà con una sola istruzione.
Nel codice mi sono limitato a dare un nome ad ogni *pane* (il che è bene fare *sempre*, soprattutto per la persistenza degli oggetti, che vedremo fra poco), una caption, e ad aggiungergli un simpatico "pin button", premendo il quale è possibile disancorarlo dai bordi. Dopo aver terminato le aggiunte, è obbligatorio richiamare la funzione membro ***Update*** dell'*auiManager*, in modo da applicare le modifiche apportate. Infine, è fondamentale deinizializzare il manager prima del termine dell'applicazione.

```
PhoneFrame::~PhoneFrame(){auiManager.UnInit();}
```

Il risultato potete ammirarlo in Figura 7, scattata nell'istante in cui l'utente, dopo aver ancorato la pulsantiera in basso sta facendo la stessa cosa col display, in alto.

PERSISTENZA DI PROSPETTIVE

Ora l'utente può scegliere una fra le infinite disposizioni dei *pane* della tastiera e del display. C'è da aspettarsi che ci spenda su un bel po' di tempo, ma quando si sarà deciso esigerà che l'applicazione ricordi le sue impostazioni, senza dover rifare tutto daccapo. Grazie a wxAui, non dovete preoccuparvi di spendere mesi ad implementare un vostro sistema di persistenza: vi basterà salvare la "prospettiva" corrente. La funzione membro *SavePerspective*, infatti, vi permette di ottenere una stringa contenente lo stato del manager, che potrete richiamare usando *LeadPerspective*.

```
void OnButton(wxCommandEvent& event) {
    wxButton* button =
      static_cast<wxButton*>(event.GetEventObject()
                                                   );

    display_->AppendText(button->GetLabelText());
    if (button->GetLabelText() == _T("*")) {
        wofstream file("impostazioni");
        wxString tmp =auiManager.SavePerspective();
        file << tmp.c_str();
    }
    if (button->GetLabelText() == _T("#")) {
        wifstream file("impostazioni");
        wstring tmp; file >> tmp;
        auiManager.LoadPerspective(tmp);
    }
}
```

In questo codice, ho abbinato *SavePerspective* e *LoadPerspective* rispettivamente ai tasti "*" e "#" del telefono, per permettervi di giocarci dinamicamente. La prassi più consueta, invece, è salvare le impostazioni immediatamente prima dell'uscita dal programma, e caricarle nel costruttore. Per la persistenza, come vedete, mi è bastato usare gli stream standard, salvando le informazioni in un file – ma se volete un sistema più elaborato, vi consiglio di studiare la classe *wxConfig*.

CONCLUSIONI

Siamo partiti da un semplice *wxTelefono* statico e, via via, ci siamo fatti strada verso soluzioni più dinamiche, fino ad arrivare ad una gestione degna degli ambienti più raffinati. L'insieme di funzionalità, controlli, e personalizzazioni offerti da *wxAui* è impressionante, e quanto visto in queste pagine può solo considerarsi un piccolo antipasto. Spero che questo breve viaggio sia servito a incoraggiarvi nella creazione di applicazioni basate su sizer e manager, e centrate sull'utente anziché sullo sviluppatore. Il nostro viaggio alla scoperta di wxWidgets continua il mese prossimo, con altri argomenti. Non mancate!

Roberto Allegra

Fig. 7: Il wxTelefono gestito da wxAui.

L'AUTORE

Il sito www.robertoallegra.it **contiene l'elenco degli articoli pubblicati in questa rubrica, con gli inevitabili approfondimenti ed errata corrige. L'e-mail dell'autore è** posta@robertoallegra.it.

SFRUTTARE TUTTE LE RISORSE

VB.NET E GLI STRUMENTI RAD CHE MICROSOFT HA RESO DISPONIBILE PER LA PROGRAMMAZIONE DEI DISPOSITIVI MOBILI RENDONO LO SVILUPPO DI UN'APPLICAZIONE UN'OPERAZIONE NON DIFFICILE. MA COME FARE PER OTTENERE LE MASSIME PRESTAZIONI?

Imparare a programmare un PPC o uno Smartphone con Visual Basic.NET permette di familiarizzare con un linguaggio già affermato nel mondo dei programmatori, ma che sta diventando sempre più importante in tutti gli ambienti di lavoro. La stessa mitica Ferrari, il non plus ultra della ricerca della perfezione ed una delle aziende più votate al miglioramento continuo del proprio prodotto, utilizza Visual Basic per sviluppare parte dei progetti necessari alla propria scuderia (http://news.com.com/Ferraris+high-tech+tune-up/2008-1012_3-5344866.html).

***Fig. 1:** L'applicazione Globular Cluster mentre viene eseguita all'interno dell'emulatore per Smartphone*

REQUISITI

Conoscenze richieste
Conoscenze di base sulla programmazione

Software
Microsoft Visual Studio.NET 2005, Windows Mobile 5.0 Smartphone SDK

Impegno

Tempo di realizzazione

PARTICOLARITÀ DELLE APPLICAZIONI MOBILI

In questo articolo imparerete a creare semplici animazioni e fare i conti con le risorse limitate di uno dei minicomputer "meno attrezzati" che ci siano in circolazione: il telefono cellulare intelligente, lo Smartphone. I dispositivi mobili in effetti diventano sempre più potenti mano a mano che vengono introdotte nuove versioni sul mercato. Tuttavia essi sono ancora molto meno potenti dei computer desktop. Le loro limitazioni hardware sono chiaramente un fattore da tenere in considerazione in fase di sviluppo di programmi.

La differenza in capacità di elaborazione tra PC desktop e device mobili è veramente rilevante. Un tipico PC ha ormai almeno 1 GHz di processore, mentre i computer più moderni superano anche 3 GHz. I Pocket PC possono essere dotati di CPU da 400 MHz e gli Smartphone solitamente hanno un processore anche meno potente. La conseguenza di ciò è che le applicazioni che contengono algoritmi complessi vengono eseguite su di essi con notevole lentezza.

Allo stesso modo il quantitativo di memoria disponibile per un'applicazione che gira su un dispositivo mobile è notevolmente limitata. Mentre un tipico PC desktop ha come minimo ormai almeno 512 MB di memoria, un altrettanto tipico smart device può disporne di un quarto in meno.

E' anche importante considerare le caratteristiche del monitor che ha un effetto veramente significativo su come un utente percepisce le prestazioni di un'applicazione. In effetti la differenza più ovvia rispetto a un PC sono le dimensioni del display. I PC possono avere anche monitor superiori a 21" mentre un tipico Pocket PC ha un piccolo display LCD magari da 6x8 centimetri o poco più. In generale la risoluzione media di uno smart device è 320x240 pixel e non supera mai i 640x480, preistoria per i monitor da PC! In questo modo chiunque percepisce subito che le applicazioni per device mobili possiedono senza dubbio meno funzionalità di quelle equivalenti per PC.

KICK-OFF DEL PROGETTO

Al solito avviate Visual Basic.NET, fate clic sul menu *File-New project*, doppio clic su *Smart Device*, clic su *Windows Mobile 5.0 Smartphone* e nella sezione *Templates* fate, poi, clic su *Device Application*. Nel campo Name digitate il nome del che volete dare al progetto, ad esempio *Globular Cluster* e fate clic sul pulsante *OK*.

Innanzitutto dovete modificare alcune proprietà predefinite della form. Ricordatevi che per attivare la finestra Properties relativa ad un oggetto qualsiasi, dovete fare un solo clic su di esso (qui la form) e, poi, scegliere dal menu *View* la voce *Properties* oppure premere direttamente il tasto F4. Le proprietà della form da aggiornare sono le seguenti:

- *(Name)=frmOra*. Si tratta qui del nome logico, vale perciò la pena di cambiare anche quello fisico e cioè il vero e proprio nome della form, visibile dal file system (Risorse del computer). Aprite con Ctrl+R il Solution Explorer, che contiene tutti i componenti del progetto corrente, fate un click destro su *Form1.vb*, scegliete l'opzione *Rename*, digitate *frmOra.vb* e premete Invio.
- *Text=Globular Cluster*. Questa è la descrizione che viene visualizzata sulla barra del titolo dell'applicazione.
- *Backcolor=Black*
- *FormFactor=Windows Mobile 5.0 Smartphone QVGA*. Questa impostazione serve a definire il tipo di emulatore che deve ospitare l'applicazione a design time. Quello selezionato garantisce una risoluzione dello schermo pari a 240x320.

Visualizzate ora la Toolbox, scegliendo dal menu *View* la voce *ToolBox* e fate doppio clic sull'oggetto *Label*, per riportarlo all'interno della *Form1*. Questo controllo visualizzerà l'ora corrente. Dopo avere fatto un solo clic sulla *Label* e avere premuto *F4*, impostate alcune proprietà come segue:

- *(Name)=lblOra*
- *Font.Size=14*
- *Font.Bold=True*
- *ForeColor= DeepSkyBlue*
- *Location.X=16*
- *Location.Y=38*
- *Size.Width=207*
- *Size.Heigth=29*
- *Text=* <cancellare il valore predefinito *Label1*>
- *TextAlign=TopCenter* <così il testo viene sempre centrato>

Siccome l'appetito vien mangiando, inserite un'altra Label che vi servirà per visualizzare la data odierna. Le proprietà che dovete impostare sono le seguenti:

- *(Name)=lblData*
- *Font.Size=10*
- *Font.Bold=True*
- *ForeColor= Aqua*
- *Location.X=2*
- *Location.Y=8*
- *Size.Width=227*
- *Size.Heigth=22*
- *Text=* <cancellare il valore predefinito *Label1*>
- *TextAlign=TopCenter*

Ora accedete al DVD allegato alla rivista e copiate la cartella Globular Cluster - Immagini sul disco del PC. Dalla Toolbox prelevate anche una *PictureBox* che ospiterà l'animazione e impostate le seguenti proprietà:

- *(Name)=picGC*
- *Image=* <fate clic sul pulsante a destra della dicitura (nessuno) e nella finestra che appare individuate la cartella con le immagini, selezionate BHo01.jpg e fate clic su *Apri*>
- *Location.X=26*
- *Location.Y=82*
- *Size.Width=188*
- *Size.Heigth=150*

Fig. 2: Le immagini che consentono di creare l'animazione

Inserite, poi, un oggetto *Timer* dalla Toolbox. Questo controllo è uno dei più potenti e utili, in quanto esegue periodicamente gli algoritmi che sono stati inseriti al suo interno. Tra l'altro, al contrario di altri controlli, viene sempre nascosto in fase di esecuzione. Le proprietà da modificare sono (*Name*) da impostare uguale a *tmrOra* e *Interval* che deve essere impostato uguale a 1000 perché l'orologio deve "girare" secondo dopo secondo. *Interval*, infatti, permette di definire la frequenza esprimendola in millisecondi, quindi 1000 millisecondi = 1 secondo. Impostate, infine, uguale a *True* la proprietà *Enabled* per garantire il funzionamento del Timer.

E' anche necessario un secondo controllo di questo tipo per permettere il funzionamento dell'animazione in maniera indipendente dall'orologio.

NOTA

IL CONTROLLO TIMER

Solo se la proprietà Enabled è uguale a True il Timer genera un evento Tick "attivo" che, cioè, viene verificato e avviato ogni x secondi, specificati nella proprietà Interval.

Inserite, dunque, un altro *Timer* che chiamerete *tmrAnima*, impostando *Interval* uguale a 200 per garantire un visualizzazione sufficientemente veloce delle animazioni (ogni 200 millesimi di secondo, cioè ogni 2 centesimi di secondo). Come al solito attivate *Enabled*, impostandola a *True*.
Se non lo avete già fatto salvate il progetto con un clic sulla sesta icona (dischetti multipli), indicate il percorso di destinazione e fate clic sul pulsante OK.

Fig. 3: Il primo salvataggio del progetto.

NOTA

LE IMMAGINI DELL'ESA

Si ringrazia l'Agenzia Spaziale Europea (http://www.esa.int) per le immagini che hanno consentito la realizzazione dell'animazione. Esse rappresentano uno zoom eseguito dal telescopio spaziale Hubble su Globular Cluster NGC 2808. Dei 150 cluster globulari conosciuti nella nostra galassia (la Via Lattea) NGC 2808 è uno dei più grandi e densi di stelle.

IL WINDOWS FORM DESIGNER

A fronte dell'inserimento di oggetti visuali dalla Toolbox Visual Basic.NET crea automaticamente delle linee di codice che corrispondono esattamente a tutti i controlli parte del progetto. Per accedere a questo codice dovete attivare l'editor del codice e selezionare nella combo box in alto a sinistra (quella che contiene gli oggetti grafici) la form. In questo caso scegliete *frmGlobularCluster* e vedrete che nella combo box di destra comparirà automaticamente (*Declarations*); selezionate al suo posto l'evento InitializeComponent che, insieme al precedente Dispose(Boolean), risulta disabilitato. A questo punto si apre una schermata che non deve essere mai modificata direttamente e che, come già detto, consente di vedere come il tool di sviluppo ha definito ed inzializzato tutti i vari elementi che voi avete inserito nel progetto, form inclusa. Ad esempio la label che contiene la data ha le caratteristiche che seguono:

```
Me.lblData.Font=NewSystem.Drawing.Font("Nina",10.
        0!, System.Drawing.FontStyle.Regular)
    Me.lblData.ForeColor =
                System.Drawing.Color.Aqua
    Me.lblData.Location = New
                System.Drawing.Point(2, 8)
    Me.lblData.Name = "lblData"
    Me.lblData.Size = New
                System.Drawing.Size(227, 22)
    Me.lblData.TextAlign =
        System.Drawing.ContentAlignment.TopCenter
```

Ricordate sempre che questi eventi possono essere aperti ed esaminati, ma è sempre meglio non modificare qui alcuna istruzione. Qualsiasi aggiornamento deve essere eseguito direttamente sugli oggetti nella sezione grafica.

Fig. 4: Il Windows Form Designer..

CREARE L'ANIMAZIONE

Le animazioni per essere veramente fluide necessitano di un numero non indifferente di frame: ogni secondo si dovrebbero avvicendare ben 25 fotogrammi! In questo modo si ha la garanzia di avere un ottimo effetto visivo, ma s'impone allo Smartphone un notevole sacrificio in termini di memoria allocata, che assume valori esponenziali mano a mano che aumenta la lunghezza temporale della nostra animazione. Nel caso d'esempio non vogliamo impegnare in maniera eccessiva le risorse di Windows Mobile, anche perché tutto sommato si tratta semplicemente di un filmato d'intrattenimento che accompagna la visualizzazione dell'orologio digitale e della data odierna. Per questo motivo caricherete e visualizzerete solo 10 fotogrammi: l'effetto visivo è ugualmente piacevole, anche se non privo d'imperfezioni.
Prima di tutto dovete inserire le dieci immagini nel progetto VB.NET. Inserite, dunque, dieci PictureBox sulla form e battezzatele rispettivamente *picGC1, picGC2, ..., picGC10*. Aggiornate ciascuna proprietà *Image* con la rispettiva immagine da *BHo01.jpg* a *BHo10.jpg*. Impostate, inoltre, tutte le proprietà *Visibile* a *False*, per non visualizzare mai in fase run time queste PictureBox.
Fate doppio clic sul secondo Timer *tmrAnima* e all'interno dell'evento Tick inserite l'algoritmo che segue.

```
Private Sub tmrAnima_Tick(...) Handles
                        tmrAnima.Tick
```

```
    Static ctr As Integer = 0
    ctr += 1
    With picGC
        Select Case ctr
            Case 1 : .Image = picGC1.Image : Case
                2 : .Image = picGC2.Image
            Case 3 : .Image = picGC3.Image : Case
                4 : .Image = picGC4.Image
            Case 5 : .Image = picGC5.Image : Case
                6 : .Image = picGC6.Image
            Case 7 : .Image = picGC7.Image : Case
                8 : .Image = picGC8.Image
            Case 9 : .Image = picGC9.Image : Case
                10 : .Image = picGC10.Image
        End Select
    End With
    If ctr = 10 Then ctr = 0
End Sub
```

Fig. 5: Le proprietà delle PictureBox

Innanzitutto notate l'utilizzo del costrutto *With...End With* che consente di portare a fattor comune un oggetto, eseguendo una serie di istruzioni su di esso senza doverlo ripetere all'interno del blocco *With*. Senza di esso avreste dovuto ripetere *picGC* dieci volte, cioè tante volte quanti sono i *Case*. Osservate anche la variabile statica *ctr*: essendo *Static* è visibile per l'intera durata del programma, ma solo all'interno dell'evento che lo ha inizializzato, cioè *tmrAnima_Tick*; al di fuori di esso non è dunque possibile utilizzare il contatore *ctr*, dove risulterebbe non dichiarato. Inoltre ricordate che *tmrAnima_Tick* viene attivato automaticamente ogni due centesimi di secondo e ad ogni suo richiamo la variabile *ctr* viene incrementata di una unità grazie all'istruzione *ctr+=1* che si potrebbe anche indicare così: *ctr=ctr+1*.
Notate, poi, i due punti che separano le istruzioni *Case*, riga dopo riga: si tratta di una sintassi concisa consentita da Visual Basic.NET che evita di dover specificare una *Case* per ogni riga; utilizzatela solo se vi sembra che il programma mantenga la sua leggibilità. Anche la *If* in linea prima della *End Sub* è una sintassi ammessa e sostituisce la seguente *If* estesa:

```
If ctr = 10 Then
    ctr = 0
End If
```

In pratica a seconda del mutare del valore della variabile *Static ctr*, che viene incrementata ad ogni ciclo di tmrAnima, l'immagine picGC (quella che abbiamo creata per prima che è anche l'unica visibile) viene inizializzata via via con le varie immagini contenute nelle altre PictureBox. L'istruzione Select Case consente di valutare qual'è il valore corrente di ctr e quale immagine deve essere visualizzata.
L'ultima *If* prima dell'*End Sub* reimposta *ctr* a zero, se il suo valore è uguale a dieci. È un'operazione necessaria per fare in modo che l'animazione ricominci dalla prima immagine *picGC1.Image*.

Fig. 6: Nell'editor del codice è presente la casella combinata che contiene tutti gli oggetti programmabili nei vari eventi

L'OROLOGIO IN FUNZIONE

Facciamo doppio clic su *tmrOra* per "entrare" automaticamente nel suo evento *Tick*. Qui dobbiamo inserire il codice che consenta di mantenere costantemente aggiornata l'ora e la data di sistema. In entrambi i casi bisogna utilizzare la funzione predefinita *Now()*, che contiene proprio data e ora di sistema. *Now()* deve, però, essere formattata, cioè manipolata per estrarre in un caso solo la data mentre nell'altro solo l'ora. A titolo esemplificativo provate ad inserire nell'evento *Tick* sia *lblData.Text* che *lblOra.Text* uguagliata solo a *Now()*.
Per ottenere la data bisogna applicare a *Now()* un'altra funzione predefinita di Visual Basic.NET cioè *Format()*, che deve ricevere due parametri. Il primo è proprio *Now()*, il secondo è *dddd - d MMM*

ISTRUZIONE SELECT CASE

Essa esegue uno dei vari blocchi di istruzioni indicati nei rami Case seguenti, sulla base del valore dell'espressione specificata proprio dopo la Select Case. Prima della End Case è possibile specificare il ramo Case Else che viene eseguito solo se nessuno dei Case precedenti è stato verificato.

UTILIZZARE IL SOFTWARE ALLEGATO

Come prima cosa copiare l'intero progetto Globular Cluster dal DVD al vostro PC. Per avviarlo all'interno di Visual Basic.NET entrate nella cartella Globular Cluster e fare doppio clic sul file Globular Cluster.sln.

yyyy, che serve a specificare che si vuole estrarre dal primo parametro una data nel formato: giorno della settimana, trattino centrale (-), numero del mese, prime tre lettere del mese ed infine anno. Per ottenere l'orario (ora, minuti e secondi) è sufficiente indicare come secondo parametro *Long Time*.

```
Private Sub tmrOra_Tick(...) Handles tmrOra.Tick
    lblData.Text = Format(Now(), "dddd - d MMM yyyy")
    lblOra.Text = Format(Now(), "Long Time")
End Sub
```

Fig. 7: Ecco come appare l'applicazione se non formattate la funzione Now().

IL MENU DI GLOBULAR CLUSTER

Per concludere degnamente l'applicazione dovete anche progettare il menu che consenta la corretta uscita dal programma e di eseguire altre eventuali attività. L'oggetto MainMenu è presente nella Toolbox, tuttavia quando si crea un nuovo progetto viene automaticamente incorporato nella form. Fintanto che non viene riempito con le voci necessarie non è visibile durante la fase di run time. Per impostarlo fare clic su mainMenu1, nella barra che contiene anche i controlli Timer, e digitare *Esci* al posto di *Type Here*. Il menu può annidarsi in sottomenu, ma in questa applicazione non è necessario. Fate clic su *Esci* e selezionate la finestra delle proprietà per modificare *(Name)* in *mnuEsci*. Fate ora clic a destra di Esci e ancora una volta compare *Type Here* dove dovete digitare *Informazioni...* Anche in questo caso impostate la proprietà *(Name)* battezzandola *mnuInfo*. Ora si tratta di programmare gli eventi del menu. Fate doppio clic su *Esci* e nell'evento *mnuEsci_Click* che appare digitate l'istruzione che consente di chiudere l'applicazione: *Application.Exit()*. Per chiedere una conferma dovete utilizzare la funzione *MsgBox()* come segue:

```
Private Sub mnuEsci_Click(...) HandlesmnuEsci.Click
    If MsgBox("Sei sicuro di voler uscire?", _
        MsgBoxStyle.YesNo Or
            MsgBoxStyle.DefaultButton2 Or _
        MsgBoxStyle.Question) = MsgBoxResult.Yes
            Then
        Application.Exit()
    End If
End Sub
```

Il primo parametro della *MsgBox()* contiene la domanda posta all'utente dell'applicazione; il secondo è un insieme di impostazioni necessarie per visualizzare i pulsanti Sì e No (*MsgBoxStyle.YesNo*), il pulsante No come predefinito (*MsgBoxStyle.DefaultButton2*) e l'icona del punto interrogativo (*MsgBoxStyle.Question*).
Infine nel menu *Informazioni...* dovete specificare le indicazioni relative a chi ha sviluppato il programma ed, eventualmente, alla versione dello stesso.

IL DEBUG DEL PROGRAMMA

Eseguire il debug di un programma significa letteralmente togliere le cimici, spulciare, cioè ripulire l'applicazione da errori di programmazione. Questa fase può essere eseguita sia sull'emulatore che direttamente sul device mobile. Un esempio classico di operazione di debug è la verifica del valore di una o più variabili: dovete premere F9 sulla riga dove si vuole che il programma si fermi e, poi dopo avere avviata la fase di run time, dovete premere Ctrl+G per visualizzare una finestra dove è possibile testare il valore delle variabili. Ad es. se si preme F9 su ctr+=1 e si esegue l'applicazione, il flusso del programma si fermerà proprio su quell'istruzione; dopo avere premuto Ctrl+G sarà possibile verificare nella finestra di Controllo immediato il valore corren-

te di ctr digitando ?ctr e poi premendo Invio.

Fig. 8: Durante la fase di debug è possibile verificare il valore delle variabili per correggere gli errori di programmazione

A volte il debug viene eseguito anche per analizzare l'esecuzione dell'applicazione passo dopo passo e per individuare qual è l'istruzione eseguita per prima. In questo caso premete F9 sull'istruzione iniziale, ad esempio *With picGC* all'interno dell'evento *tmrAnima_Tick* e, poi, eseguire l'applicazione con F5, o all'interno dell'emulatore o dello Smartphone. Il programma si fermerà appena raggiunta l'istruzione evidenziata con F9 e potrete andare avanti passo dopo passo con il tasto F8 (o menu *Debug-Step Into*). Ricordate che, nel caso, siano presenti eventi asincroni, come quelli generati dal controllo *Timer*, non sarà possibile seguire facilmente il flusso del programma step by step e si potranno verificare dei risultati imprevedibili.

ESECUZIONE E DISTRIBUIRE L'APPLICAZIONE

A questo punto dovete abbinare il progetto ad un'icona: accedete al menu *Project-Globular Cluster Properties* e scegliete a sinistra la scheda *Application*. Nel campo *Icon* dell'applicazione fate clic a destra sul pulsante con i puntini, selezionate l'icona star.ico (presente nel DVD nella cartella Globular Cluster - Immagini).
Per eseguire l'applicazione premete F5 e scegliete *Windows 2005 Smartphone Device* se volete visualizzarla direttamente nello Smartphone, oppure *Windows 2005 Smartphone Emulator* per attivarla all'interno dello Smartphone virtuale che l'ambiente di sviluppo ci mette a disposizione.
Per distribuire l'applicazione, una volta ultimata, fate clic sul menu *Build* e, poi, su *Build Globular Cluster* per creare la versione definitiva del file eseguibile. Copiate, quindi, il file Globular

Fig. 9: Per impostare l'icona bisogna accedere alle proprietà del progetto.

Cluster.exe, presente in Globular Cluster\bin\Release, all'interno dello Smartphone (in Globular Cluster\bin\Debug è presente l'eseguibile che viene generato tutte le volte che si esegue il programma nell'emulatore). Nessun altro file sarà necessario perché Windows Mobile 2005 contiene già tutte le librerie indispensabili al funzionamento delle applicazioni .NET.

Enrico Bottari

Fig. 10: L'applicazione in esecuzione in uno Smartphone i-mate SP5.

GESTIONE DEI DATI ATTRAVERSO JDBC

LA RAPPRESENTAZIONE DELL'INFORMAZIONE È SOLO UNO DEI TANTI ASPETTI NELLA PROGETTAZIONE DI UN'APPLICAZIONE. UN RUOLO DI FONDAMENTALE IMPORTANZA VIENE RIVESTITO DALL'INTERFACCIAMENTO CON I DATABASE. ECCO COME FARE...

L'applicazione che andremo a sviluppare in questo articolo riguarda una semplice gestione di libri, il catalogo di una possibile biblioteca. Le informazioni che dovremo gestire sul database sono essenzialmente due, gli utenti e i libri. La prima tabella ci servirà per gestire una sorta di autenticazione e registrazione che deve essere fatta per accedere alla nostra applicazione. La seconda tabella invece servirà semplicemente per catalogare tutte le entry relative ai libri. L'approccio che utilizzeremo per la realizzazione di questa Web Application e soprattutto per la gestione dei bean è abbastanza semplice. Definiremo due bean, uno per l'utente e uno per il libro, che verranno utilizzati da diversi managed bean che sono direttamente collegati alle pagine con dei form. In questo modo avremo due diverse tipologie di bean. La prima è quella semplice, dove vengono inseriti i vari attributi che rispettano lo schema del database e dove vengono creati i vari metodi get/set. La seconda tipologia invece riguarda i bean che gestiscono i campi dei vari form, dove andremo ad inserire alcuni metodi per validare le informazioni, dove verranno inserite le action da eseguire nel caso in cui i campi inseriti nei vari form siano corretti. È proprio quest'ultimo tipo di bean, che dovranno gestire i semplici bean per il mapping. Questo sarà possibile attraverso delle classi DAO (Data Access Object), che ci permetteranno in maniera molto semplici di dialogare con il database utilizzando le classi API JDBC.

IL DATABASE

Come database utilizzeremo in questo caso MySQL, dovremo configurare nella nostra applicazione i driver JDBC relativi. La scelta del database non è vincolante, soprattutto se non vengono utilizzate le librerie specifiche dei vari produttori ma ci si appoggia direttamente alle API JDBC standard. Lo schema è abbastanza semplice, perché non dobbiamo gestire nessuna relazione tra le due entità. Di seguito trovate le definizioni che abbiamo utilizzato per la creare queste due tabelle sul database MySQL

UTENTI	
PK	USERNAME
	PASSWORD EMAIL

LIBRI	
PK	ISBN
	AUTORE TITOLO DESCRIZIONE PRESENTE

Fig. 1: Schema del nostro database

```
CREATE SCHEMA `biblioteca`;
CREATE TABLE `biblioteca`.`UTENTI` (
  `USERNAME` VARCHAR(45) NOT NULL,
  `PASSWORD` VARCHAR(45) NOT NULL,
  `EMAIL` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`USERNAME`)
)
ENGINE = InnoDB;
CREATE TABLE `biblioteca`.`LIBRI` (
  `ISBN` VARCHAR(45) NOT NULL,
  `TITOLO` VARCHAR(45) NOT NULL,
  `AUTORE` VARCHAR(45) NOT NULL,
  `DESCRIZIONE` VARCHAR(45) NOT NULL,
  `PRESENTE` BOOLEAN NOT NULL,
  `PREZZO` DOUBLE NOT NULL,
  PRIMARY KEY (`ISBN`)
)
ENGINE = InnoDB;
```

I BEAN PER IL MAPPING E LE CLASSI DAO

Dobbiamo ora definire due diversi bean per mappare le informazioni e le relative classi

DAO che ci permetteranno di gestire queste informazioni e caricarle/salvarle sul database attraverso l'uso dei bean. Il primo bean che troviamo qui di seguito è quello relativo alla definizione dell'utente

```
package ioprogrammo.jsf.database;
public class User {
    private String username;
    private String password;
    private String email;
    public User() {
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

La classe DAO che si occuperà di gestire le informazioni dell'utente è UserDAO, dove sono stati definiti dei semplici metodi statici che saranno poi richiamati dalla nostra applicazione JSF. Qui di seguito trovate il codice relativo alla classe UserDAO, dove vengono utilizzate le classiche API JDBC.

```
package ioprogrammo.jsf.database;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class UserDAO {
    private static String
        URI="jdbc:mysql://127.0.0.1:3306/biblioteca";
    private static String LOGIN="root";
    private static String PASSWD="mysql";
    public UserDAO() {
    }
    public static boolean isRegistered(String
                                        username) {
        boolean found=false;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn =
DriverManager.getConnection(URI, LOGIN, PASSWD);
            PreparedStatement
        ps=conn.prepareStatement("SELECT USERNAME
FROM BIBLIOTECA.UTENTI WHERE USERNAME=? ");
            ps.setString(1,username);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                found=true;
            }
            ps.close();
            conn.close();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
        return found;
    }
    public static String register(User user) {
        String register="";
        boolean insert=false;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn =
DriverManager.getConnection(URI, LOGIN, PASSWD);
            PreparedStatement
                ps=conn.prepareStatement("INSERT INTO
BIBLIOTECA.UTENTI(USERNAME,PASSWORD,EMAIL)
                                  VALUE(?,?,?)");
            ps.setString(1,user.getUsername());
            ps.setString(2,user.getPassword());
            ps.setString(3,user.getEmail());
            int res = ps.executeUpdate();
            if (res==1) {
                register="ok";
                insert=true;
            }
            else
                register="Utente non trovato";
            ps.close();
            conn.close();
        } catch (Exception ex) {
            if (!insert)
                register=ex.toString();
            System.out.println(ex.toString());
        }
        return register;
    }
    public static String login(User user) {
        String login="";
        boolean found=false;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn =
DriverManager.getConnection(URI, LOGIN, PASSWD);
            PreparedStatement
        ps=conn.prepareStatement("SELECT USERNAME
```

```
FROM BIBLIOTECA.UTENTI WHERE USERNAME=?
                        AND PASSWORD=?");
            ps.setString(1,user.getUsername());
            ps.setString(2,user.getPassword());
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                found=true;
            }
            if (found)
                login="ok";
            else
                login="Utente non trovato";
            ps.close();
            conn.close();
        } catch (Exception ex) {
            if (!found)
                login=ex.toString();
            System.out.println(ex.toString());
        }
        return login;
    }
}
```

In questa classe DAO abbiamo definito 3 metodi che ci serviranno nella nostra applicazione JSF, login(), register() e isRegistered(). Il primo verrà utilizzato per effettuare il login nella nostra applicazione, quindi in base allo username e alla password che sono stati forniti ed inseriti all'interno del bean User, verifichiamo se questo utente è presente nel nostro database. Il secondo metodo serve invece per effettuare la registrazione dell'utente e isRegistered() verrà utilizzato per controllare la presenza o meno dello username nel nostro database, visto che proprio lo username è la chiave della nostra tabella. La classe Libro invece serve come bean per rappresentare i vari libri che dovremo gestire e quindi devono essere inserite tutti gli attributi e metodi get/set relativi per gestire l'informazione relativa al libro

```
package ioprogrammo.jsf.database;
public class Libro {
    private String titolo;
    private String autore;
    private String descrizione;
    private boolean presente;
    private String isbn;
    public Libro() {
    }
    public String getTitolo() {
        return titolo;
    }
    public void setTitolo(String titolo) {
        this.titolo = titolo;
    }
    public String getAutore() {
        return autore;
    }
    public void setAutore(String autore) {
        this.autore = autore;
    }
    public String getDescrizione() {
        return descrizione;
    }
    public void setDescrizione(String descrizione) {
        this.descrizione = descrizione;
    }
    public boolean isPresente() {
        return presente;
    }
    public boolean getPresente() {
        return presente;
    }
    public void setPresente(boolean presente) {
        this.presente = presente;
    }
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
}
```

In questo caso la classe DAO che utilizzeremo è LibroDAO, che ha 3 diversi metodi: insert(), alreadySaved() e retrieveAll(). I primi due servono rispettivamente per inserire un nuovo libro e per sapere se un libro è già stato inserito, prendendo come chiave l'ISBN (International Standard Book Number). Il metodo retrieveAll() ci servirà invece per caricare nella nostra applicazione tutti i libri presenti e visualizzarli poi tramite le componenti UI di JSF

```
public static ArrayList retrieveAll() {
        ArrayList lista=new ArrayList();
        Libro libro;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn =
    DriverManager.getConnection(URI, LOGIN, PASSWD);
            Statement
                statement=conn.createStatement();
            ResultSet
                rs=statement.executeQuery("SELECT
    ISBN,TITOLO,AUTORE,DESCRIZIONE,PRESENTE
                                FROM LIBRI");
            while(rs.next()) {
                libro=new Libro();
                libro.setIsbn(rs.getString("ISBN"));
                libro.setTitolo(rs.getString("TITOLO"));
```

```
            libro.setAutore(rs.getString("AUTORE"));
libro.setDescrizione(rs.getString("DESCRIZIONE"));
libro.setPresente(rs.getBoolean("PRESENTE"));
            lista.add(libro);
        }
        rs.close();
        statement.close();
        conn.close();
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
    return lista;
}
```

REGISTRAZIONE

Passiamo ora a definire le varie componenti che serviranno per gestire la registrazione dell'utente. La pagina principale della nostra applicazione, index.jsp, deve semplicemente riportare i due diversi link per la registrazione e per il login dell'utente, perché vogliamo che solo utenti registrati possano accedere ad essa.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
    <head>
        <meta http-equiv="Content-Type"
                content="text/html; charset=UTF-8">
        <title>Biblioteca</title>
    </head>
    <body>
    <center>
    <img src='logo.png'>
    <table border='1'>
        <tr>
            <td>
                <h3>Registrazione</h3>
                <a
 href="./faces/register.jsp">Registrazione utente</a>
            </td>
            <td>
                <h3>Login</h3>
                <a href="./faces/login.jsp">Effettua il
                                                login</a>
            </td>
        </tr>
    </table>
    </center>
    </body>
</html>
```

Nella pagina relativa alla registrazione dovremo gestire il classico form, dove saranno presenti i 3 diversi campi richiesti, ovvero username, password ed email. Abbiamo bisogno che tutti e tre i campi siano riempiti quindi inseriremo l'attributo required="true" all'interno dei diversi campi di input

Registrazione	Login
Registrazione utente	Effettua il login

Fig. 7: Pagina principale

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="f"
          uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h"
          uri="http://java.sun.com/jsf/html"%>
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title>Registration page</title>
    </head>
    <body>
        <center>
            <img src='logo.png'>
            <f:view>
            <h:form id="register-form">
            <h1><h:outputText value="Register"
                                        /></h1>
             <h:message for="register-form"/><br>
            <h:outputLabel value="Username:"
                                for="username"/>
            <h:inputText required="true"
                                id="username"
              validator="#{register.validateUsername}"
                  binding="#{register.username}" />
            <h:message for="username"/><br>
            <h:outputLabel value="Password:"
                                for="password"/>
            <h:inputSecret required="true"
     id="password" binding="#{register.password}" />
            <h:message for="password"/><br>
            <h:outputLabel value="Email:"
                                   for="email"/>
            <h:inputText required="true" id="email"
                validator="#{register.validateEmail}"
                    binding="#{register.email}" />
            <h:message for="email"/><br><br>
            <h:panelGroup>
                <h:commandButton value="Register"
                    action="#{register.register}" />
                <h:commandButton value="Cancel"
```

```
                    immediate="true" action="exit" />
            </h:panelGroup>
            </h:form>
        </f:view>
        </center>
    </body>
</html>
```

Come è possibile vedere, i campi username ed email hanno specificato un validator, che è legato ad un bean che gestisce la registrazione, che vedremo qui di seguito. Se i valori non rispettano la validazione che viene gestita da questi metodi, verrà ripresentato il form con accanto al campo di input un messaggio d'errore. Questo form viene gestito dal managed bean Register, che oltre ad avere definite le componenti UI che abbiamo utilizzato e due diversi metodi per la validazione, ha anche il metodo register() che serve ad effettuare la vera e propria registrazione

```
package ioprogrammo.jsf.database;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.component.html.HtmlInputSecret;
import javax.faces.component.html.HtmlInputText;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;
import javax.servlet.http.HttpSession;
public class Register {
    private HtmlInputText username;
    private HtmlInputSecret password;
    private HtmlInputText email;
    public Register() {
    }
    public void validateUsername(FacesContext
                        context, UIComponent toValidate,
            Object value) throws ValidatorException {
        String username = (String) value;
        if (UserDAO.isRegistered(username)) {
            FacesMessage message = new
                FacesMessage("Username già utilizzato");
            throw new ValidatorException(message);
        }
    }
    public void validateEmail(FacesContext context,
                            UIComponent toValidate,
            Object value) throws ValidatorException {
        String email = (String) value;
        if (!email.matches(".+@.+\\.[a-z]+")) {
          ((UIInput)toValidate).setValid(false);
          throw new ValidatorException(new
                FacesMessage("Email non valida"));
        }
    }
    public String register() {
        User user=new User();
user.setUsername(username.getValue().toString());
user.setPassword(password.getValue().toString());
        user.setEmail(email.getValue().toString());
        String register=UserDAO.register(user);
        if (register.equals("ok")) {
            FacesContext fc =
javax.faces.context.FacesContext.getCurrentInstance(
                                                        );
            Object
object=fc.getApplication().getVariableResolver().resol
                            veVariable(fc,"user");
            User userS=(User)object;
            userS.setUsername(user.getUsername());
            return "registered";
        }
        else {
FacesContext.getCurrentInstance().addMessage("regi
    ster-form",new FacesMessage("Registration error:
                                    "+register));
            return "no-registered";
        }
    }
    public HtmlInputText getUsername() {
        return username;
    }
    public void setUsername(HtmlInputText username)
                                                    {
        this.username = username;
    }
    public HtmlInputSecret getPassword() {
        return password;
    }
    public void setPassword(HtmlInputSecret
                                        password) {
        this.password = password;
    }
    public HtmlInputText getEmail() {
        return email;
    }
    public void setEmail(HtmlInputText email) {
        this.email = email;
    }
}
```

Vediamo quindi cosa viene fatto nel dettaglio dalla classe Register. Il metodo validateUsername() è quello che viene utilizzato per la validazione del campo username del form di registrazione. Questo metodo controlla attraverso il metodo isRegistered() di UserDAO se l'utente è già presente all'interno del database. In caso positivo viene sollevata una ValidatorException che invierà al form di registrazione un determinato messaggio d'errore. Il metodo validateEmail() ha un comportamento analogo, solo che in questo caso viene controllato il campo email utilizzando

ALTERNATIVE

Invece dell'approccio JDBC che è stato utilizzato nell'articolo per realizzare le classi DAO, è possibile utilizzare dei framework opensource di persistenza come
Hibernate http://www.hibernate.org/
iBatis http://ibatis.apache.org/
db4o http://www.db4o.com/
pBeans http://pbeans.sourceforge.net/
JPOX http://www.jpox.org/

una semplice espressione regolare che definisce la classica email (ovvero stringa@stringa.stringa). Abbiamo infine il metodo register(), quello che viene richiamato da JSF nel caso in cui tutte le informazioni sono state inserite correttamente. In questo metodo non facciamo altro che creare un bean User e passarlo come argomento al metodo register() di UserDAO. Nel caso in cui la registrazione viene effettuata senza problemi il bean User viene inserito nella sessione della nostra Web Application e si procede verso il menu (attraverso una navigation rule che vedremo definita in seguito), altrimenti viene segnalato all'utente l'errore che si è verificato.

LOGIN

Il login della nostra applicazione funziona in maniera simile a quella vista precedentemente per la registrazione. Abbiamo quindi una pagina JSF con un form per l'inserimento dati, un managed bean che gestisce questo form e che utilizza il DAO definito per la classe User ed un metodo che decide se il login è stato effettuato con successo. Vediamo quindi la pagina login.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="f"
          uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h"
          uri="http://java.sun.com/jsf/html"%>
<html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Login Page</title>
  </head>
  <body>
    <center>
      <img src='logo.png'>
      <f:view>
      <h:form id="login-form">
      <h1><h:outputText value="Login" /></h1>
       <h:message for="login-form"/><br>
      <h:outputLabel value="Username:"
                     for="username"/>
      <h:inputText required="true"
   id="username" binding="#{login.username}" />
      <h:message for="username"/><br>
      <h:outputLabel value="Password:"
                     for="password"/>
      <h:inputSecret required="true"
    id="password" binding="#{login.password}" />
      <h:message for="password"/><br><br>
      <h:panelGroup>
```

```
          <h:commandButton value="Login"
                   action="#{login.login}" />
          <h:commandButton value="Cancel"
               immediate="true" action="exit" />
        </h:panelGroup>
        </h:form>
      </f:view>
      </center>
    </body>
</html>
```

In questo caso abbiamo due soli componenti UI, che servono per l'inserimento dello username e della password. In questo caso il managed bean che ci permette di gestire questo form è Login, attraverso il quale possiamo effettuare il vero e proprio login nel sistema utilizzando il metodo login() che viene richiamato dal nostro form quando i dati sono stati inseriti

Fig. 3: Pagina di login

```
package ioprogrammo.jsf.database;
import javax.faces.application.FacesMessage;
import javax.faces.component.html.HtmlInputSecret;
import javax.faces.component.html.HtmlInputText;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;
public class Login {
  private HtmlInputText username;
  private HtmlInputSecret password;
  public Login() {
  }
  public String login() {
    User user=new User();
user.setUsername(username.getValue().toString());
user.setPassword(password.getValue().toString());
    String login=UserDAO.login(user);
    if (login.equals("ok")) {
      FacesContext fc =
javax.faces.context.FacesContext.getCurrentInstance(
);
      HttpSession session =
((HttpSession)fc.getExternalContext().getSession(fals
e));
      Object
    object=fc.getApplication().getVariableResolver().
                   resolveVariable(fc,"user");
```

```
            User userS=(User)object;
            userS.setUsername(user.getUsername());
            return "logged";
        }
        else {
FacesContext.getCurrentInstance().addMessage("logi
    n-form",new FacesMessage("Login error: "+login));
            return "no-logged";
        }
    }
    public String logout() {
        FacesContext fc =
javax.faces.context.FacesContext.getCurrentInstance(
                                                    );
        HttpSession session =
((HttpSession)fc.getExternalContext().getSession(fals
                                                e));
        session.invalidate();
        Object
object=fc.getApplication().getVariableResolver().resol
                                veVariable(fc,"user");
        User userS=(User)object;
        userS=null;
        return "exit";
    }
    public HtmlInputText getUsername() {
        return username;
    }
    public void setUsername(HtmlInputText username)
                                                    {
        this.username = username;
    }
    public HtmlInputSecret getPassword() {
        return password;
    }
    public void setPassword(HtmlInputSecret
                                        password) {
        this.password = password;
    }
}
```

Il metodo login() utilizza UserDAO (e il relativo metodo login()) per capire se l'utente può accedere o meno. In caso positivo, il bean User viene registrato nella sessione e si passa direttamente al menu della nostra applicazione. Come potete vedere è stato definito anche un metodo logout(), che ci permetterà successivamente di far uscire il nostro utente dall'applicazione. Dopo aver effettuato il login (oppure dopo aver completato la registrazione) veniamo rediretti verso la pagina menu.jsp, dove vengono elencati tre semplici link che possiamo cliccare, l'aggiunta di un libro, la lista dei libri e il logout

```
<f:view>
        <img src='logo.png'>
        <h3>Ciao, <h:outputText
    value="#{user.username}" /></h3><br><br>
        <h:form>
            <a href="aggiungiLibro.jsp">Inserisci
                                    libro</a><br>
            <a href="lista.jsp">Consulta
                    biblioteca</a><br><br><br>
            <h:commandButton rendered="true"
        action="#{login.logout}" value="Esci" />
        </h:form>
</f:view>
```

GESTIONE DEI LIBRI

Siamo giunti quindi alla parte relativa ai libri. Nella prima pagina, aggiungiLibro.jsp, dobbiamo realizzare il classico form che servirà poi per salvare un nuovo libro sul database. Anche in questo caso abbiamo collegato i vari campi del form ad un managed bean, LibroInsert, che richiama il metodo insert() di LibroDAO per salvare un nuovo libro sul database. Qui di seguito vengono riportati i metodi significativi della classe LibroInsert

```
public void validateISBN(FacesContext context,
                            UIComponent toValidate,
        Object value) throws ValidatorException {
    String isbn = (String) value;
    if (LibroDAO.alreadySaved(isbn)) {
        FacesMessage message = new
                    FacesMessage("Libro già salvato");
        throw new ValidatorException(message);
    }
}
public String insert() {
    Libro libro=new Libro();
    libro.setIsbn(isbn.getValue().toString());
    libro.setAutore(autore.getValue().toString());
libro.setDescrizione(descrizione.getValue().toString())
                                                    ;
    libro.setTitolo(titolo.getValue().toString());
    libro.setPresente(presente.isSelected());
    String register=LibroDAO.insert(libro);
    if (register.equals("ok")) {
        return "inserted";
    }
    else {
FacesContext.getCurrentInstance().addMessage("inse
        rt-form",new FacesMessage("Insert error:
                                    "+register));
        return "no-inserted";
    }
}
```

Il metodo validateISBN serve per la validazione dell'input relativo al form d'inserimento del nuovo libro. Questo, come nel caso della registra-

zione dell'utente, controlla la presenza di libro con lo stesso ISBN (chiave della tabella LIBRI) e in caso positivo lancia una ValidatorException. Il metodo insert() invece si prende carico del salvataggio del libro, creando un nuovo oggetto Libro e passandolo alla relativa classe DAO che si occuperà della vera e propria query SQL per l'inserimento sul database. Ora per completare la nostra applicazione dobbiamo rappresentare la lista dei vari libri in una tabella. Per recuperare tutti i libri dal database utilizziamo un semplice managed bean, che viene utilizzato per avere un ArrayList dei libri e che viene richiamato dalla nostra pagina JSP.

```
package ioprogrammo.jsf.database;
import java.util.ArrayList;
public class ListaLibri {
    private ArrayList libri;
    public ListaLibri() {
    }
    public ArrayList getLibri() {
        libri=LibroDAO.retrieveAll();
        return libri;
    }
    public void setLibri(ArrayList libri) {
        this.libri = libri;
    }
}
```

Infine dobbiamo creare la pagina JSP, che utilizzerà il componente UIData di JSF per mostrare a schermo una semplice tabella con tutti i libri da noi inseriti

```
<h:dataTable id="listalibri" value="#{lista.libri}"
        var="libro" border="1">
    <h:column>
        <f:facet name="header">
            <h:outputText
                value="Autore"/>
        </f:facet>
        <h:outputText
            value="#{libro.autore}" />
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText
                value="Titolo"/>
        </f:facet>
        <h:outputText
            value="#{libro.titolo}" />
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText
                value="ISBN"/>
        </f:facet>
        <h:outputText
            value="#{libro.isbn}" />
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText
                value="Descrizione"/>

        </f:facet>
        <h:outputText
        value="#{libro.descrizione}" />
    </h:column>
    <h:column>
        <f:facet name="header">
        <h:outputText
            value="Presente"/>
        </f:facet>
        <h:selectBooleanCheckbox
    disabled="true" value="#{libro.presente}" />
    </h:column>
</h:dataTable>
```

L'ArrayList, che viene richiamato grazie alla classe ListaLibri, viene gestito come in un ciclo for, dove vengono stampate a schermo tutte le informazioni relative al libro, contenuto nel-

Fig. 4: Lista dei libri inseriti nella nostra applicazione

l'oggetto Libro che viene caricato dal database.

CONCLUSIONI

Abbiamo visto uno dei tanti modi in cui è possibile gestire i bean che utilizziamo all'interno della nostra applicazione JSF, come salvarli e caricarli in memoria utilizzando delle classi DAO che fanno da ponte tra la nostra applicazione e il database. L'applicazione può essere chiaramente migliorata, permettendo magari la modifica e la cancellazione dei libri inseriti. In questo caso l'approccio che è stato utilizzato si è basato sul classico JDBC, ma un'altra opzione, molto valida, è quella relativa a dei framework per la persistenza, che possono facilitare di molto la nostra vita, evitando la creazione di classi DAO con complessi metodi.

Federico Paparoni

L'AUTORE

L'autore, Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

DIAGRAMMI DI STATO

ABBIAMO IMPARATO COME MODELLARE UN'APPLICAZIONE E A RAPPRESENTARE IL FLUSSO DELLE OPERAZIONI. POSSIAMO ADESSO INDENTIFICARE IN CHE CONDIZIONI SI TROVA L'APPLICAZIONE IN UN PRECISO ISTANTE? LA RISPOSTA È SI. ECCO COME

In questa puntata conclusiva del corso su UML affronteremo altre due tipologie di diagrammi molto importanti: i diagrammi di macchina a stato e quelli di deploy. I primi consentono di modellare l'insieme di stati in cui può trovarsi un'applicazione o un progetto, definendo gli eventi che portano alla transizione da uno stato all'altro. Il secondo tipo di diagramma consente di rappresentare la configurazione fisica dei computer su cui gira l'applicazione, e non solo. Infatti, UML non è limitato alla progettazione della struttura interna dell'applicazione e al suo funzionamento, ma si preoccupa anche degli elementi fisici di supporto alla stessa.

DIAGRAMMI DI MACCHINA A STATO

Tutte le applicazioni per computer che utilizziamo possiedono una gestione dello stato, anche se in alcuni casi questo potrebbe non essere evidente per via della semplicità delle funzionalità implementate dal software. Un esempio di applicazione basata su stati potrebbe essere il tipico sistema di e-commerce, dove si possono identificare i seguenti stati (semplificati):

1. In attesa di login da parte dell'utente. In questo caso il sistema è in modalità "vetrina" e le funzioni di check-out sono disabilitate.
2. In fase di shopping. Qui il sistema supporta la funzione di "carrello degli acquisti" e la funzione di check-out e ordini passati sono attive.
3. In fase di ordine. L'utente ha selezionato il check-out e il sistema sta chiedendo i dati di conferma, come le modalità di spedizione e pagamento.

Ovviamente, è possibile che il punto 3 contenga un ulteriore insieme di stati, per esempio per mantenere traccia della pagina di dati che l'utente sta lavorando in un dato momento, nel caso il processo di check-out sia composto da una serie di schermate successive.

Si noti che un diagramma di macchina a stati, sebbene simile, non è la stessa cosa di un diagramma di attività, descritto nell'articolo precedente. Le attività sono sequenze di cose che vengono fatte dall'utente, dal sistema e modellano un processo. Un diagramma degli stati individua invece gli stati in cui si può trovare l'applicazione per soddisfare le attività presenti nel primo tipo di diagrammi. Le due tipologie di modelli rappresentano quindi informazioni che tra loro sono trasversali. Solitamente uno stato identifica anche un'insieme di funzioni attive e disattive. Per esempio, si potrebbe semplificare e dire che Microsoft Word ha due stati: un documento è aperto per modifica, oppure nessun file è aperto. Nel primo stato sono attive una serie di funzioni, come quelle di salvataggio, copia, ecc.; nel secondo queste non sono attive, perché non hanno senso per lo stato attuale.

DISEGNARE DIAGRAMMI DI MACCHINA A STATO

Per produrre i diagrammi UML di questo articolo utilizzeremo come nelle puntate precedente lo strumento open source Argo UML (http://argouml.tigris.org). Per creare un nuovo diagramma di macchina a stato, in inglese Statechart Diagram, procedere come segue:

1. avviare l'applicazione
2. selezionare la funzione New Statechart Diagram sotto il menu Create

Il programma presenta l'usuale finestra di progettazione, con una barra di strumenti specifica per il disegno di questo tipo di diagrammi (**Figura 1**). Vedremo tra breve come utilizzare gli elementi più importanti.

***Fig. 1:** Barra degli strumenti per la creazione di diagrammi di macchina a stati.*

Un diagramma di macchina a stati è caratterizzato, in modo similare ai diagrammi delle attività, da un nodo iniziale e uno finale. L'aspetto grafico di questi elementi è uniforme a quelli dei diagrammi di attività: rispettivamente sono rappresentati da un pallino pieno e un pallino vuoto con attorno un cerchio. Anche i singoli stati ricordano molto un'attività, in quanto sono rappresentati da riquadri dagli angoli arrotondati, con la differenza che gli stati sono caratterizzati da una linea orizzontale che suddivide a metà il riquadro. Nella parte superiore c'è il nome dello stato, che lo identifica univocamente e ne fornisce una prima indicazione semantica. Si possono scorgere le icone relative a questi elementi in Figura 1: lo stato è la terza icona da sinistra, mentre l'inizio e la fine sono il nono e il decimo pulsante.
Come si ricorderà dai numeri precedenti, per inserire un nuovo elemento nel diagramma è sufficiente fare clic sull'elemento desiderato, per prelevarlo, e poi fare clic sull'area di disegno, per posizionarlo nel punto desiderato.

UN PRIMO ESEMPIO

Vediamo ora come disegnare il semplice diagramma presente in **Figura 2**. Rappresenta gli stati che può assumere una lampadina collegata a un interruttore, una situazione che sarà familiare a tutti. Una lampadina può essere solo in due stati: accesa o spenta. Come si nota dalla Figura 2, infatti, sono presenti solo due riquadri, che rappresentano appunto questi due stati.

***Fig. 2:** Un semplice esempio di diagramma di macchina a stati.*

Per indicare che da uno stato è possibile passare al successivo si utilizzano linee che terminano con una punta di freccia aperta. La presenza di quest'ultimo elemento permette di capire il verso della transizione. Nella Figura 2 si evince che dallo stato Acceso si può passare allo stato Spento e viceversa. Inoltre, si può notare la presenza di una descrizione sulla freccia, che solitamente è l'indicazione dell'azione che porta alla transizione di stato. Per esempio, per passare da Acceso a Spento è stata utilizzata la funzione "spegni", che in questo contesto equivale al premere l'interruttore della luce. Si sarebbe potuto anche scrivere questa frase più prolissa, dipende dal contesto per il quale è stato disegnato il diagramma. Se è destinato a utenti, funzionali o in generale deve utilizzare una visione ad alto livello, conterrà termini e descrizioni più generali e discorsive. Se invece è destinato ad analisti e programmatori, probabilmente avrà più a che fare con elementi tecnici e con il codice.
Vediamo come disegnare questo diagramma. Per prima cosa è necessario inserire gli elementi in Figura 2 come prima descritto. In secondo luogo è necessario tirare le frecce di transizione, cosa che è possibile realizzare in due modi:

1. Facendo clic su un elemento del diagramma. A fianco di questo appaiono icone di frecce, che è possibile trascinare su un altro elemento del diagramma stesso.
2. Facendo clic sull'icona della transizione nella barra degli strumenti (quinta icona da sinistra), facendo clic sull'elemento grafico di partenza e poi trascinando la linea su quello di destinazione.

Per inserire il nome dello stato nel riquadro è possibile procedere in due modi:

1. Facendo doppio clic sulla parte superiore del riquadro. In quella posizione viene visualizzato un campo di testo che è possibile compilare a piacimento.
2. Facendo un singolo clic sullo stato. Nel pannello Properties vengono riportati i dati dell'elemento scelto. Qui è possibile indicare il nome dello stato nel campo Name.

Si ricorda invece che la creazione di angoli nelle linee di transizione avviene trascinando un punto qualsiasi della linea stessa.
Per inserire la descrizione sulla linea, invece, è possibile procedere in due modi diversi:

- Facendo doppio clic sulla linea. Viene presentato un campo di testo sulla linea stessa, in cui è possibile specificare l'evento (trigger) che produce il cambio di stato.
- Facendo un clic con il tasto destro sulla linea e selezionando New Trigger dal menu contestuale che viene visualizzato. Da qui è possibile scegliere tra quattro tipologie di trigger. Quella più utilizzata è il Call Event.

Una transizione può essere caratterizzata, oltre che da un evento, da una guardia. Questa rappresenta una condizione, che deve essere verificata

NOTA

SUL WEB

Per una introduzione facile e veloce alla modellazione UML, anche se nella sua più recente versione 2.0, su consulti il sito www.agilemodeling.com, mantenuto da un noto esperto del settore, Scott Ambler.

perché si possa passare allo stato di destinazione. Questa espressione può essere scritta in linguaggio naturale, codice nel linguaggio di programmazione preferito oppure in OCL (Object Constraint Language). Quest'ultimo è un linguaggio per l'espressione di vincoli inserito nelle specifiche UML2. Una guardia è racchiusa tra parentesi quadre.

STATI DI GIUNZIONE

Ma i diagrammi di macchina a stato possono essere utilizzati anche per definire gli stati che può assumere un oggetto all'interno di un'applicazione. Per esempio, si consideri un'ipotetica applicazione per gestire le richieste di autorizzazione di un ufficio pubblico, per esempio per l'utilizzo di strutture comuni. Questa potrebbe gestire un insieme di stati associati alla richiesta stessa:

- creata (appena sottoposta all'ufficio);
- in lavorazione (l'impiegato smista all'ufficio competente);
- in valutazione (i grandi pensatori stanno decidendo);
- approvata (la richiesta può essere soddisfatta);
- respinta (c'è qualche motivo per cui la richiesta non è accoglibile).

Un diagramma di macchina a stati che può rappresentare questa situazione è illustrato in Figura 3. Come si può vedere, ci sono una sequenza di stati e transizioni di passaggio da uno al successivo. In uscita dallo stato di valutazione, però, è presente uno stato di giunzione, rappresentato da un rombo. Da qui è possibile creare una biforcazione per gestire i due diversi esiti della valutazione. Infatti dal rombo escono due transizioni, ciascuna delle quali porta a uno stato specifico. Da questi due stati poi si arriva allo stato finale.

Fig. 3: I diagrammi di macchina a stati possono.

Quando in un diagramma di macchina a stati è presente uno stato di giunzione è necessario specificare una guardia che indichi la logica per cui si dovrebbe transitare verso uno stato piuttosto che verso l'altro. Questo è indispensabile, altrimenti non si saprebbe dove andare. In questo caso le guardie sono espresse in linguaggio naturale. Per inserirle è sufficiente fare clic sulla linea di transizione e poi selezionare l'icona [G] presente nella barra degli strumenti. Digitare quindi nel campo expression del pannello Properties la descrizione della condizione. Opzionalmente è possibile indicare un nome per la guardia, ma questo non appare nel diagramma, serve solo a poter riutilizzare la stessa in un altro contesto, cercandola negli elementi di progetto.

STATI COMPOSITI

Un'altra caratteristica interessante di diagammi di macchina a stati è la possibilità di definire stati compositi. Questi particolari stati sono costituiti da ulteriori sottostati. Gli stati compositi sono rappresentati da un riquadro di stato decisamente più grande del normale, all'interno del quale è possibile inserire più sotto-stati che definiscono un livello di dettaglio maggiore. In Figura 4 è presente un esempio che è una rielaborazione dell'argomento precedente, ma focalizzato sulla fase di valutazione. Come si nota, questa volta la richiesta viene creata e subito dopo passa nello

Fig. 4: I diagrammi di macchina a stati possono.

stato "In valutazione".
Questo stato è composto da diversi sottostati, che indificano le varie fasi di valutazione. In questa ipotesi, se l'esito di uno stato non è soddisfatto, non si passa al successivo. Quindi per prima cosa si passa alla fase di verifica della solvibilità: all'interno di questo stato ci saranno una serie di richieste di documentazione per capire se chi ha fatto la richiesta può pagare il compenso di noleggio pattuito. In caso positivo si passa alla fase di verifica antimafia: qui vengono richiesti i documenti necessari quando si opera con una struttura pubblica (sono necessari per gli appalti, il nostro esempio è puramente ipotetico). Quindi c'è la fase di intervista, anch'essa caratterizzata dalle sue attività specifiche.
Il diagramma termina qui, ma si sarebbe potuto concludere con l'invio di un messaggio di conferma o di diniego a chi ha inoltrato la richiesta.

NOTA

UML E PATTERN

Se interessa approfondire UML e il suo rapporto con i Design Pattern è possibile consultare il seguente testo: Massimiliano Bigatti, "UML e Design Pattern. Notazioni grafiche e soluzioni per la progettazione", Hoepli Editore, ISBN: 978-88-203-3657-8.

DIAGRAMMI DI DEPLOY

Il termine deploy affonda le sue radici in contesto militare, dove veniva utilizzato per descrivere il posizionamento di attrezzature e truppe nel campo di battaglia. Nel contesto informatico ha assunto il significato di installazione, collaudo e implementazione di un sistema basato su computer o di un'applicazione. Potrebbero essere oggetto di deploy una nuova rete per l'azienda, l'installazione di un server farm, l'implementazione di una nuova applicazione in un ambiente di programmazione distribuita.

Con i diagrammi di deploy è possibile disegnare la struttura fisica di un'applicazione distribuita, locale, di una configurazione di rete e così via. In sostanza, è possibile rappresentare le singole tipologie di hardware richiesto e i componenti fisici che realizzano il programma. Questi possono essere applicazioni, programmi, file eseguibili, file di configurazione, ecc.

In un diagramma di deploy gli elementi fisici su cui può girare un software sono rappresentati da parallelepipedi e sono chiamati nodi. Questi possono essere computer, dispositivi integrati, palmari, smartphone, ma anche firewall hardware o altri elementi simili. In generale, qualsiasi elemento fisico pertinente e necessario per il diagramma che si sta disegnando.

Gli elementi software sono invece rappresentati da riquadri dei componenti e sono solitamente inseriti nei nodi o in altri componenti.

Per creare un nuovo diagramma di deploy in ArgoUML si seleziona la funzione New Deployment Diagram nel menu Create. La relativa barra degli strumenti è abbastanza semplice e consente di creare nodi, componenti e descrivere relazioni. Oltre a queste funzioni sono presenti quelle basilari di ArgoUML, già viste nelle barre degli strumenti di tutti gli altri diagrammi e che consentono l'inserimento di elementi grafici standard e di note.

In Figura 5 è presente un esempio di diagramma che descrive in modo non eccessivamente particolareggiato una classica architettura per un'applicazione web. È stato creato utilizzando gli strumenti presenti nella barra nel modo usuale. Unico particolare: per inserire un elemento dentro un altro è sufficiente trascinare il primo sull'ultimo. A sinistra si può vedere il nodo Server, in altre parole, un computer preposto al ruolo di servente. Al suo interno troviamo due componenti: il Web Server e il Database. La presenza di questi due elementi all'interno del nodo server sta ad indicare che sul computer girano due processi: il server web e il server di database. Ovviamente, sul computer girano anche altri processi, altrimenti questo non funzionerà affatto. Si ricordi che la logica generale dei diagrammi UML è quella di rappresentare i soli elementi essenziali e necessari per il contesto che si sta trattando. All'interno del Web Server gira l'applicazione web (per esempio un'applicazione scritta in PHP).

Il nodo a destra rappresenta invece il computer Client, al cui interno gira il browser.

Fig. 5: Un esempio di diagramma di deploy.

Questo diagramma è molto semplice, ma sarebbe potuto essere anche più complicato. Per esempio, se l'applicazione avesse fatto uso di un'Applet, si sarebbe potuto indicare la sua presenza in esecuzione sul browser e come archivio sul file system del Web Server. Tutto dipende dall'accento che si vuole dare al diagramma.

Si noti inoltre la freccia di dipendenza che unisce il Client al Server: consente di indicare che il Client utilizza il Server. Sarebbe anche stato possibile unire i due nodi con una linea continua, sulla quale riportare il protocollo di comunicazione utilizzato. In questo caso sarebbe stato html/http. La linea continua identifica un legame più stretto e in questo caso sarebbe potuto essere anche appropriato.

Un altro aspetto interessante dei diagrammi di deploy è che consentono di specificare, all'interno dei riquadri dei componenti o nei nodi, una stringa di proprietà che consente di specificare meglio eventuali requisiti. Per esempio, si potrebbe indicare che il Server deve per forza utilizzare Linux, magari perché l'applicazione sfrutta alcune caratteristiche specifiche. In questo caso si può aggiungere, sotto il nome del nodo, una stringa simile a {os=Linux}. Come si nota, le proprietà sono identificate da coppie chiave valore racchiuse tra parentesi graffe.

CONCLUSIONI

Con questo articolo si conclude questo breve corso di UML, di cui sono stati affrontati i diagrammi più importanti che è possibile realizzare con un utile strumento open source quale ArgoUML.

Massimiliano Bigatti

NOTA

BIBLIOTECA

Un libro decisamente economico ma che rappresenta un buon riferimento per tutti gli elementi di UML 2.0 è il seguente: Enrico Amedeo "UML Pocket", Editore Apogeo, ISBN: 978-88-503-2627-3.

INFRASTRUTTURA DEI SISTEMI VIRTUALI

I NOSTRI GIORNI VEDONO LA PRESENZA DI UN ELEVATO NUMERO DI SISTEMI OPERATIVI DISPERSI IN DIFFERENTI VERSIONI. PER NON INCORRERE IN PROBLEMI È IMPORTANTE TESTARE IL NOSTRO SOFTWARE SU PIÙ TIPOLOGIE DI INSTALLAZIONE, MA COME FARE?

Conoscenze richieste

Conoscenze di base dei sistemi operativi

Software

Una versione aggiornata di Vmware

Impegno

Tempo di realizzazione

Oggigiorno, i responsabili di un CED, sono più che mai "costretti" a tranquillizzare le rispettive aziende sulle capacità delle proprie infrastrutture IT circa la loro capacità di soddisfare le esigenze di business in maniera "economicamente" efficace. Questo obiettivo è certamente molto arduo da raggiungere ed è "rallentato" dal tentativo di cercare di ottenere gli stessi risultati con un numero inferiore di server nelle sale CED. Il motivo di tali preoccupazioni è soprattutto economico, poiché, all'aumentare dei server presenti in azienda, aumentano le spese totali legate ad un maggior consumo di corrente, alla manutenzione dell'hardware, ecc.

Tutte queste problematiche, legate alle predette necessità, hanno portato alla ribalta il fenomeno della *virtualizzazione*, una tecnologia che da diversi anni incontra l'interesse di molte aziende e che tutt'oggi incuriosisce molti utenti. In quest'articolo affronteremo il tema in maniera soprattutto pratica, cercando di fare una panoramica sui principali concetti legati a questo mondo e a come VMWare consenta, con il suo Virtual Infrastructure 3, di realizzare un'architettura "virtuale" al di sopra di qualunque aspettativa.

COS'È LA VIRTUALIZZAZIONE?

Prima di procedere oltre, è doveroso cercare di capire cosa s'intende con il termine *virtualizzazione*. Anche rischiando di entrare nella banalità e di non usare una terminologia propria di queste tecnologie, cercheremo di spiegare questo termine a chi si avvicina ad esse per la prima volta.

Immaginiamo innanzitutto di avere un certo numero di server, diciamo una decina e di volerli consolidare in un'unica macchina, lasciandone inalterate le caratteristiche principali (indirizzo IP, servizi, ecc.). Attraverso la virtualizzazione, tutto ciò è possibile ed il risultato finale è quello di avere gli stessi benefici/servizi offerti dalle 10 macchine, ma con 10 "pezzi di ferro" in meno. Tutto quello che accade al termine di questa "migrazione" è quello di avere un unico server all'interno del quale girano 10 oggetti ciascuno dei quali rappresenta uno dei predetti server. Dal punto di vista dell'utente è tutto traparente, perché le macchine virtuali continueranno a svolgere il loro lavoro servendosi delle risorse del server che le ospita.

Questo modo di rappresentare lo scenario che vede protagonisti le macchine virtualizzate e l'host che le ospita, è volutamente reso banale, ma dovrebbe rendere meglio l'idea del risultato finale che si ottiene.

Quando parliamo di ambienti con centinaia di server, è evidente che il processo di migrazione da server fisico a server virtuale, necessità di particolari accorgimenti e che, a monte, è necessario dotarsi di un apposito prodotto che consenta il raggiungimento di questo risultato. Nel nostro articolo prenderemo in considerazione il pacchetto di programmi studiati ad hoc per queste operazioni ed offerte da VMWare.

La suite Virtual Infrastructure di VMWare è costituita essenzialmente da 3 prodotti principali:

- **Virtual Center**: costituisce il collettore delle informazioni rilevate sui singoli host ESX. Attraverso esso è possibile ottenere informazioni su di essi, sulle virtual machine configurate su ogni ESX oltre permettere di configurare ogni singolo aspetto dell'intera infrastruttura.
- **ESX Server**: rappresenta, fisicamente, l'host sul quale "risiedono" le virtual machine. In realtà, le cose non stanno esattamente così nel senso che lo storage sul quale è possibile installare e configurare ogni singola macchina virtuale, può essere locale all'ESX oppure, ad esempio, far riferimento ad una SAN.
- **License Server**: attraverso questo modulo vengono gestite le licenze relative all'intera infrastruttura.

Per avere un'idea ancor più chiara di quanto si nasconda dietro questa serie di prodotti, ecco qualche accenno ad altri importanti concetti che, come riportato dai manuali ufficiali, costituiscono le componenti essenziali della Virtual Infrastructure:

- **VirtualCenter Management Server**: è il nodo di controllo per configurare, implementare e gestire gli ambienti virtualizzati. Il *Management Server* è un servizio installato su sistema operativo Windows 2000, Windows XP Professional o Windows Server 2003.
- **VirtualCenter Database**: utilizzato per memorizzare le informazioni persistenti sui server fisici, i resource pool e le virtual machine gestite da VirtualCenter *Management Server*. Il database che è possibile utilizzare può essere Oracle, Microsoft SQL Server o MSDE.
- **Virtual Infrastructure Client**: permette agli amministratori di connettersi remotamente al *VirtualCenter Management Server* o direttamente sui server ESX da un qualsiasi PC con sistema operativo Windows.
- **VirtualCenter Agent**: installato sui server ESX, permette la connessione da parte di *VirtualCenter Management Server*.
- **Virtual Infrastructure Web access**: permette la gestione delle VM e l'accesso alla console grafica delle VM senza la necessità di installare un client.

A questo punto, dopo una brevissima infarinatura sui principali termini legati alla virtualizzazione secondo VMWare, cominciamo il nostro cammino verso la costruzione di un'ambiente virtuale.

L'INSTALLAZIONE DI VIRTUAL CENTER

L'installazione di Virtual Center è caratterizzata da tre step principali:

- Configurazione del DB che ospiterà le informazioni.
- Installazione della componente server di *Virtual Center* e del relativo *License Server* per la gestione delle licenze.
- Installazione del *Virtual Infrastructure Client* necessario alle operazioni di configurazione dei server ESX e del server Virtual Center stesso.

Il primo passo da portare a termine, quindi, è l'installazione e configurazione del database, cuore delle informazioni che via via andranno a popolare questa base dati. Virtual Center 2, come menzionato poco fa, supporta Oracle, SQL Server e Microsoft MSDE. Facendo riferimento al solo caso di SQL Server, tutto quello che occorre fare, una volta definita una nuova istanza del DB in SQL Server e assicurati che tutte le tabelle siano riposte al suo interno, è quello di creare un nuovo SQL *Server ODBC Connection* inserendo poche semplici informazioni che "riportano" all'istanza SQL prima creata. Una volta che questo passo è stato ultimato, possiamo finalmente avviare l'installazione di Virtual Center (possibile purtroppo solo su macchine Windows).
Innanzitutto, assicuriamoci che la macchina che ospiterà Virtual Center, risponda ai seguenti prerequisiti:

- **Sistema operativo**: Windows 2000 Server SP4 with Update Rollup 1, Windows 2003 o Windows XP Professional.
- **Processore**: 2.0GHz or superiore Intel or AMD x86.
- **RAM**: almeno 2GB di RAM.
- **Spazio disco**: Almeno 560MB di spazio disco libero (2GB consigliati), tenendo presente che, nel caso il DB sia locale, occorre una quantità superiore di spazio oltre a quella consigliata.

Appurato questo, inseriamo il CD d'installazione e clicchiamo sulla voce **VirtualCenter Management Server**. Nel caso fosse necessario, ci verrà richiesto d'installare anche *Microsoft .NET Framework version 1.1*. Seguiamo le richieste del wizard e completiamo l'installazione.

Figura 1: *La schermata che appare inserendo il CD d'installazione di Virtual Center*

A questo punto, considerato che abbiamo a disposizione il CD già inserito, clicchiamo sulla seconda voce della schermata principale, *Virtual Infrastructure Client*, che ci permetterà d'installare il tool per il collegamento non solo al Virtual Center, ma anche ai singoli host ESX. Tralasciamo per semplicità i dettagli di questo passaggio e accenniamo ora all'installazione del *License Server*.
Anche qui valgono le stesse considerazioni fatte poc'anzi. Non occorrono consigli particolari: è sufficiente cliccare sulla voce *License Server* per dare inizio all'installazione di quest'altro componente.
Terminate queste prime tre fasi, è il momento di dedicarci all'installazione di un host ESX, il cuore dell'infrastruttura sul quale si poggiano le singole macchine virtuali.

INSTALLAZIONE DI UN HOST ESX

Innanzitutto, così come fatto per Virtual Center, occorre assicurarsi che l'hardware sul quale risiederà questa componente, sia sufficientemente "dotato" per consentirne l'installazione. Ecco i prerequisiti richiesti:

- **Processore**: almeno 2 processori di questo tipo:

  o 1500MHz Intel Xeon and later, or AMD Opteron (32Å]bit mode);

  o 1500MHz Intel Viiv or AMD A64 x2 dual_core processors;

- **RAM**: almeno 1GB RAM.
- **Rete**: una o più schede Ethernet.
- **Disco**: uno SCSI disk, Fibre Channel LUN o RAID LUN con spazio non partizionato.

Appurato anche in questo caso di possedere l'hardware necessario a supportare il prodotto e verificata la connettività di rete, avviamo il nostro server con il CD d'installazione di ESX Server 3.01 inserito. Per semplicità, considereremo solo il caso di una configurazione con storage locale, tralasciando gli altri casi.
Appena dopo il caricamento del CD, ci viene mostrata una schermata che ci chiede in che maniera procedere (fase *GUI* o *TEXT* dell'installazione). Diamo *Invio* per iniziare quella a modalità grafica. Tralasciando i dettagli sull'installazione del mouse e "amenità" simili, clicchiamo su *Install* (in luogo di *Upgrade*) quando ci verrà richiesto. Considerando che stiamo trattando l'installazione di un host ESX nuovo di zecca, possiamo procedere seguendo le impostazioni predefinite (laddove possibile) ed inserendo quelle "personali" (come l'indirizzo IP statico e la password dell'utente *root*) quando richiesto.
Al termine dell'installazione, il nostro ESX è pronto ad ospitare le VM. Non resta quindi che collegarci al singolo ESX o al Virtual Center tramite il Virtual Infrastructure Client per iniziare a "smanettare". In questa parte tralasciamo per motivi di spazio la parte relativa al licensing, rimandando questi dettagli sulla configurazione delle licenze alla documentazione VMWare. Pertanto, daremo per scontato che i server siano stati correttamente licenziati.

CREAZIONE DI UNA VIRTUAL MACHINE

Cominciamo col dire che solitamente, quando si decide di virtualizzare un certo numero di server, l'operazione che deve essere compiuta è quella di convertire una macchina fisica già presente ed operativa in una macchina virtuale. Quest'operazione può essere compiuta attraverso l'apposito tool gratuito offerto da VMWare e denominato *Converter*, oppure utilizzando appositi prodotti di terze parti a pagamento, come PowerConverter di PlateSpin.
In questo articolo, non ci addentreremo in queste operazioni, ma focalizzeremo la nostra attenzione su alcuni concetti importanti del mondo VMWare facendo vedere brevemente come costruire un macchina virtuale da zero.
Avviamo dunque il client della Virtual Infrastructure e colleghiamoci al Virtual Center fornendo l'indirizzo del server e le informazioni sulle credenziali. Se non abbiamo commesso errori, dopo pochi istanti si aprirà l'interfaccia principale del prodotto che ci permetterà di compiere tutte le operazioni necessarie.

Figura 2: Aggiunta di un host ESX a Virtual Center

Clicchiamo ora sul pulsante *Inventory* e scegliamo *Host & Clusters*. Poi clicchiamo ancora una

volta sulla voce Host & Clusters della root dell'albero mostrato nel primo frame di sinistra e scegliamo *New Datacenter*. I datacenter sono contenitori logici per host ESX, Virtual machine, ecc. e ci servono per avere una visione globale di tutto più leggibile. Fatto questo, diamo il nome che vogliamo a questa nodo dell'albero e confermiamo. Ora quello che occorre fare è inserire, all'interno del Virtual Center, gli host ESX che lo stesso dovrà gestire. Clicchiamo sulla voce del datacenter appena creato con il tasto destro del mouse e selezioniamo *Add host*. Inseriamo i dati sul nostro ESX per collegarci ad esso ed aggiungerlo nell'elenco degli host da gestire. Ovviamente ripetiamo l'operazione per tutti gli host ESX da inserire.
Adesso siamo pronti per creare una nuova Virtual Machine. L'operazione è, di per sé, molto banale. selezioniamo dal menu principale la voce *File*, poi *New* ed infine *Virtual Machine*. Questi passi faranno partire un wizard tramite il quale potremo inserire le informazioni sulla VM che si sta creando, come il nome, la memoria, il numero di schede di rete, ecc. Selezioniamo quindi l'opzione *Typical* e procediamo inserendo i dati necessari. Si noti, in particolare, che questa fase non viene installato il sistema operativo, operazione questa che dovremo compiere al termine della configurazione del wizard.
Come "suggerito" dai manuali di VMWare, una virtual machine appena creata è esattamente identica ad una nuova macchina fisica con un disco rigido non formattato.
Inseriamo dunque il CD del sistema operativo all'interno del lettore e, dopo aver modificato opportunamente le impostazioni della VM appena creata (attraverso la voce *Settings*) in maniera tale da farle utilizzare il CDROM dell'host fisico, avviamola (tasto destro sulla VM e selezione della voce *Power On*).
In questa fase, vedremo l'avvio del BIOS della macchina e, successivamente, quello del setup del sistema operativo da installare.
Da questo momento in poi, l'installazione è analoga a quella che effettueremo se avessimo una macchina "reale".
Al termine, la macchina virtuale si comporterà esattamente come la sua "sorella" fisica: potremo raggiungerla via RDP oppure effettuare un ping ad essa, spegnerla o accenderla, ecc. In poche parole: la dovremo trattare come una macchina reale.
In aggiunta a queste poche considerazioni possiamo anche dire che ogni macchina può essere facilmente clonata con pochi click di mouse o, addirittura, ne possiamo creare alcune come template da utilizzare come "base" per le prossime creazioni.

Figura 3: Una macchina virtuale vista dal Virtual Infrastructure Client collegato ad un ESX

CONCLUSIONI

In questa brevissima panoramica abbiamo mostrato in cosa possa consistere un ambiente virtuale secondo i canoni ed i vincoli offerti da VMWare. Ovviamente sarebbe comodo se tutto fosse sempre così semplice. In realtà, infatti, sono stati volutamente tralasciati alcuni "oggetti" e concetti specifici di questo prodotto, semplicemente perché si voleva focalizzare l'attenzione più sulla logica e sulle operazioni che sottintendono la virtualizzazione che sul prodotto specifico.
VMWare introduce diversi concetti per consentire un corretto uso delle risorse come i resource pool, ad esempio.
Inoltre, mediante moduli aggiuntivi, si può fare in modo da bilanciare il carico di lavoro in maniera automatica sugli host ESX coinvolti o, addirittura, prevedere lo spostamento delle VM (che in fondo altro non sono che una serie di file) su un altro host ESX in caso di failure. Insomma: l'argomento è certamente molto vasto e non può essere certamente esaurito in poche pagine.
Ogni nuovo concetto, anche solo menzionato, ci costringerebbe a dilungarci ulteriormente. Pertanto, chiunque fosse interessato all'argomento, in special modo al prodotto offerto da VMWare, troverà sul sito di questa azienda tutta la documentazione necessaria a muoversi in questo ambiente.

Francesco Lippo

SOFTWARE SUL CD

Zend Studio 5.5.0

L'EDITOR "UFFICIALE" PER PHP

PHP è uno straordinario linguaggio di programmazione che ha dalla sua parte una completezza fuori dal comune, una curva di apprendimento bassissima, una velocità d'esecuzione non facile da trovare in un linguaggio interpretato.Normalmente si può scrivere codice php persino utilizzando il notepad, ma ovviamente utilizzare un editor con caratteristiche avanzate migliora di molto la velocità di coding di qualunque programmatore.Zend Studio è un editor decisamente completo prodotto dalla software house che più di ogni altra sponsorizza lo sviluppo di PHP. Le funzionalità inserite sono tantissime, dalla navigazione fra le classi, al code complexion, alla sintax highliting, al tracing e al debugging dell'applicazione.Utilizzare questo strumento migliora facilita enormente la stesura del codice a qualunque programmatore.

Directory:ZendStudio-5_5_0a

APACHE 2.2.4

IL WEB SERVER PIÙ USATO AL MONDO

L'ultima versione del Web Server progettato da Apache e che da solo tiene in piedi una buona parte di Internet. Nonostante l'agguerrita concorrenza Apache rimane un Web Server incredibilmente usato. I suoi moduli sono praticamente illimitati, è incredibilmente leggero,viene utilizzato praticamente su tutte le piattaforme di hosting al mondo. Se avete in mente di progettare un'applicazione Web non potete fare a meno di avere installato sulla vostra macchina in locale una versione di Apache. Quella che vi presentiamo è la versione 2.3 della serie 2, che ormai è sufficientemente consolidata da essere usata in ambienti di produzione.

Directory:apache_2.2.3-win32-x86-no_ssl

IRRLICHT 1.2

ACCENDI IL MIGLIORE MOTORE 3D OPEN SOURCE!

Irrlicht è un motore per la grafica tridimensionale, scritto in C++ e utilizzabile sia con questo linguaggio, sia con la tecnologia .NET.Presenta le principali caratteristiche di un motore professionale e vanta una notevole comunità di sviluppatori, con diversi progetti in attivo. Irrlicht ha tra i suoi pregi anche quello di potere utilizzare sia DirectX che OpenGL per cui diventa particolarmente adatto anche per lo sviluppo di giochi multipiattaforma

Directory:irrlicht-1.2.zip

VISUALWX 087.6

IL RAD PER LE WXWIDGETS

Le WxWidgets sono una straordinaria libreria che consente di sviluppare applicazioni grafiche multipiattaforma in C++ con pochissimo sforzo. Unica pecca era fino a ieri l'assenza di un ambiente RAD per la costruzione delle form. Questa limitazione è adesso superata, infatti grazie VisualWX è ora

possibile disegnare con pochi colpi di mouse la propria applicazione per poi riempire di "contenuti" gli eventi associati ai vari elementi presenti sulla form.Si tratta di una comodità non da poco se si pensa a quanto facile sia sviluppare per windows e linux contemporaneamente grazie proprio alle wxwidgets

Directory:VisualWx_087-60.exe

BAKEFILE 0.2.2

IL GENERATORE DI MAKE

In molti conosceranno l'utility "make". Si tratta semplicemente di un programmino che prende in pasto un file di testo e compila un software sulla base delle indicazioni in esso contenute, linkando le corrette

librerie e utilizzando un preciso ordine. Bakefile si sposa proprio con l'utility make. Prende in pasto infatti un file di descrizione e genera il corrispondente makefile. La novità è che bakefile è in grado di generare l'output corretto per un vastissimo numero di linguaggi. In sostanza basta creare il file di descrizione per ottenere il makefile corretto per il proprio linguaggio di programmazione

Directory:bakefile-0.2.2-setup.exe

MINGGW DEVELOPER STUDIO 2.0.5

L'EDITOR PROFESSIONALE PER GCC

GCC è uno splendido compilatore, veloce, affidabile e completo. Mancava fino ad oggi un editor che lo supportasse. Questa lacuna è colmata proprio da MingGW. Un ottimo prodotto che integra perfettamente tutte le caratteristiche di GCC. Assolutamente da provare

Directory:MinGWStudioFullSetupPlus-2_05.exe

CODEBLOCK 2007

L'EDITOR MULTIPIATTAFORMA PER C++

Code::Blocks è un IDE opensource crossplatform. La particolarità di questo IDE è quello di essere modulare e basato su plugin. Supporta cioè una moltidutidne di linguaggi semplicemente aggiungendo il supporto corretto. Attualmente è orientato verso C++ ma con poco sforzo è utilizzabile proficuamente con ogni altro linguaggio/compilatore

Directory:/boo

MINIMALISTIC GNU FOR WINDOWS 5.1.3

GNU EVERYWHERE

Una collezione completa di header file e librerie che vengono combinate con gli strumenti tipici GNU per produrre programmi nativi Windows utilizzando le librerie e i compilatori della famiglia GNU

Directory:MinGW-5.1.3.exe

JAVA SE DEVELOPMENT KIT 6

IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

Directory:jdk-6-windows-i586

TOMCAT 6.0.9

IL SERVLET CONTAINER PER JAVA E JSP

L'idea è molto semplice. Sviluppare im Java pagine Web. Ad un primo sguardo, Tomcat potrebbe sembrare un normale Web Server. Ed in effetti è un normale Web Server! In grado di soddisfare le richieste per qualunque pagina HTML. In realtà però Tomcat è anche qualcosa in più, ovvero la capacità di soddisfare le richieste per applicazioni Java. Potrebbe sembrare complesso, in realtà lo è meno di quanto sembri. Immaginate Tomcat come un grande contenitore al cui interno ci sono altri contenitori ciascuno dei quali rappresenta un'applicazione Java, che una volta richiamata costruisce una pagina Web interpretabile da un browser. Questo consente di sviluppare pagine Web utilizzando tutta la potenza della normale gerarchia delle classi Java e la sintassi e il linguaggio che qualunque programmatore Java conosce bene.

Directory:apache-tomcat-6.0.9.exe

ECLIPSE SDK 3.2.2

L'IDE TUTTOFARE

Eclipse è un progetto completo portato avanti da Eclipse Foundation con la

collaborazione di una miriade di aziende fra cui IBM, Adobe, Sun e che si è prefissata lo scopo di creare un IDE estendibile per plugin adattabile a qualunque tipo di linguaggio o tecnologia. Di default Eclipse si propone come IDE per Java ed è qui che da il meglio di se. Ma proprio grazie ai suoi plugin è possibile utilizzarlo come ambiente di programmazione per PHP, per C++, per Flex e per molti altri linguaggi ancora. Inoltre sempre grazie per ciascun linguaggio sono disponibili altri plugin ad esempio per rendere l'ambiente RAD o per favorire lo sviluppo dei Web Services o altro. Insomma lo scopo è stato raggiunto completamente. Eclipse è realmente un IDE tuttofare, ormai maturo, e che serve una miriade di programmatori grazie alle sue caratteristiche di affidabilità e flessibilità. Unica nota negativa: una certa pesantezza che lo rende idoneo ad essere usato solo su PC con una dotazione hardware minima di tutto rispetto

Directory:eclipse-SDK-3.2.1-win32.zip

PHP 5.2.1

IL LINGUAGGIO DI SCRIPTING PIÙ AMATO DEL WEB

Sono tre le colonne portanti di Internet: PHP, APACHE e MySQL. Certo la concorrenza è forte. Asp.NET e SQL Server

avanzano con celerità, ma a tutt'oggi non si può affermare che i siti sviluppati in PHP costituiscano la stragrande maggioranza di Internet. Quali sono le ragioni del successo di cotanto linguaggio? Prima di tutto la completezza. PHP ha di base tutto quello che serve ad un buon programmatore, raramente è necessario ricorrere a librerie esterne, e quando è proprio indispensabile farlo esistono comunque una serie di repository che rendono tutto immediatamente disponibile ed in forma gratuita. Il secondo punto di forza del linguaggio sta nella sua capacità di poter essere utilizzato sia in modo procedurale che nella sua forma ad oggetti certamente più potente e completa. Esiste un terzo di punta di forza essenziale che è quello riguardante la curva di apprendimento. PHP è in assoluto uno dei linguaggi con la curva di apprendimento più bassa nel panorama degli strumenti di programmazione. Si tratta perciò di uno strumento indispensabile per chi si avvvicina alla programmazione web, a meno che non intendiate scegliere strade diverse quali possono essere ASP.NET o JSP

Directory:php-5.2.0-Win32.zip

ZEND FRAMEWORK 0.9

L'SDK EVOLUTO PER PHP

Chi sviluppa in PHP è abituato a sviluppare in maniera autonoma il proprio framework, partendo dalle proprie esperienze e necessità. Questo ha però dato origine ad un ecosistema di applicazioni spesso difficilmente manutenibile. Zend ha sviluppato un proprio Framework completo che dispone di una serie di meccanismi che velocizzano la risoluzione dei problemi più frequenti. Si va dall'implementazione del pattern MVC alla creazione dei Web Services, alla gestione delle stampe in PDF. Si tratta di un framework piuttosto affidabile essendo sviluppato da Zend che è anche la software house promotrice dello sviluppo di PHP.

Directory:ZendFramework-0.9.1-Beta

ZEND CORE 0

SISTEMA SEMPRE PERFETTO

Il tuo sistema è abbastanza aggiornato? hai l'ultima versione di PHP con tutte le

patch? Le applicazioni che girano sul tuo server soffrono di qualche bug di sicurezza?A tutto questo e a molto di più risponde Zend Core.Utilizzando questo prodotto sarete sempre certi di quale versione state utilizzando e della consistenza del vostro sistema. Si tratta di un tool veramente straordinario che consente ad ogni sistemista di dormire notti tranquille

Directory:ZendCore-v2.0.3-Windows-x86.zip

MYSQL 5.1.15

IL PRINCIPE DEI DATABASE

Indispensabile per programmare webapplication in tecnologia PHP. Nonche non sia possibile utilizzare altridatabase, ma MySQL e PHP rappresentano veramente un binomioinscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti

Directory:mysql-5.0.27-win32.zip

PYTHON 2.5

L'EX GIOVANE RAMPANTE

Python è stato considerato per lungo tempo il nuovo che avanza. Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione inun gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet. Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows

Directory:python-2.5.msi

POSTGRES 8.2.3

IL GRATIS COMPLETO E VELOCE

Nessun server di database offre la completezza delle funzioni esposte da PostgreSQL e rimane comunque completamente Gratis. PostgreSQL è probabilmente il più estendibile fra i database esistenti. Inutileparlare della gamma praticamente completa delle sue funzioni. Il punto di forza che lo pone probabilmente al di sopra di tutti i concorrenti rimane l'alta possibilità di personalizzazione, oltre, naturalmente, alla velocità, alla stabilità ed al costo nullo.Unica pecca, una certa complessità nella gestione. E' sicuramente da usare in ambienti di produzione professionali che non possono accontentarsi di alcun compromesso

Directory:postgresql-8.2.1-1.zip

PROGRAMMAZIONE DELLE ATTIVITÀ

DOVETE RAGGIUNGERE UN OBIETTIVO. PER FARLO AVETE NECESSITÀ DI ORGANIZZARE UNA SERIE DI COMPITI ED ASSEGNARLI A DIVERSE PERSONE. CI SONO MILIONI DI MODI DI RAGGIUNGERE LO SCOPO. MA QUAL È QUELLO PIÙ EFFICACE?

Il tempo è una variabile che riveste un ruolo da protagonista nello scheduling. Si vuole compiere un lavoro, eventualmente composto da più attività e utilizzando diverse risorse, umane o macchine, nel minor tempo possibile. È questa una prima definizione degli obiettivi che si prefiggono gli sviluppatori di algoritmi di scheduling. Tale definizione come vedremo potrà essere ampliata e approfondita giacché sotto questa etichetta si riconosce un'ampia casistica di quesiti di ottimizzazione combinatoria costruiti per la ricerca di soluzioni ottime per problemi di programmazione di lavori. Nel presente articolo proveremo a trasmettere un esempio classico e ben circoscritto di scheduling, fornendo per esso le basi metodologiche per risolverlo. Nella seconda parte, invece, dopo aver esposto una serie di altri esempi, tenteremo nello spazio che abbiamo a disposizione l'arduo compito di sintetizzare lo stato dell'arte, affrontando terminologia e classificazione, in considerazione degli enormi progressi e degli innumerevoli contributi registrati in questi ultimi quaranta anni nella suddetta area di studio.

N.	Attività elementare	Durata giorni
1	Rilevamento geologico, progettazione e preparazione cantiere	30
2	Costruzione fondamenta	12
3	Costruzione struttura (pilastri e solai)	15
4	Erezione muratura vano 1	1
5	Erezione muratura vano 2	2
6	Erezione muratura vano 3	1
7	Costruzione massetto e posa pavimentazione vano 1	2
8	Costruzione massetto e posa pavimentazione vano 1	2
9	Costruzione massetto e posa pavimentazione vano 1	2
10	Fabbricazione copertura	4
11	Intonacatura pareti vano 1	2
12	Intonacatura pareti vano 2	2
13	Intonacatura pareti vano 3	2
14	Posa infissi e porte vano 1	1
15	Posa infissi e porte vano 2	1
16	Posa infissi e porte vano 3	1
17	Pittura vano 1	1
18	Pittura vano 2	2
19	Pittura vano 3	1
20	Pulizia casa	2
21	Consegna	1

COSTRUIAMO UNA CASETTA

Per introdurci allo scheduling facciamo riferimento all'edilizia con un semplice esempio. Consideriamo il caso in cui si debba edificare una piccola costruzione di tre vani. Si possono individuare delle attività atomiche, nel senso che devono essere eseguite per intero, ognuna riferita ad un codice numerico (numerazione crescente) e ad una durata, che nel caso specifico può essere opportunamente espressa in giorni. In un primo momento è necessario specificare le singole fasi che costituiscono l'intera attività (attività elementari), un modo per farlo è produrre una tabella dove siano riportati i dati utili allo scopo, come esposto di seguito.
Esistono dei vincoli di precedenza, ovvero alcune fasi devono necessariamente essere eseguite dopo altre, ad esempio la costruzione della struttura di pilastri e solai non può che essere realizzata dopo la costruzione delle fondamenta. Cosicché l'attività n° 2 è propedeutica alla n° 3. Un valido metodo per rappresentare la gestione dei lavori del presente caso prevede l'uso di un grafo aciclico. I nodi del grafo conterranno le informazioni delle singole attività, ossia il numero di riferimento e la durata, gli archi di congiunzione modellano il concetto di precedenza, come mostrato in **figura 1**, dove è stato costruito il grafo per l'esempio proposto.
La costruzione del grafo impone in questa tipologia di problemi la programmazione del lavoro, così bisognerà decidere i lavori che vanno serializzati, ossia posti uno dopo l'altro e quelli che invece possono essere svolti in "parallelo". Nel caso specifico, come si può dedurre dal grafo esistono diverse attività, attinenti i tre vani che si vogliono costruire, che possono essere rea-

Fig. 1: "grafo corrispondente al problema di programmazione dei lavori di un cantiere per la costruzione di una piccola casa."

lizzate in contemporanea, basta avere la mano d'opera sufficiente; si tratta della pavimentazione, della pittura ed di altri. Inoltre, bisogna tener conto che il convergere di più archi su un solo nodo, implica che la sua realizzazione può essere solo successiva alla terminazione delle attività dei nodi che in esso immettono archi. Ad esempio, la copertura si può fare solo dopo la costruzione dei pilastri e dei muri. È evidente che il tempo totale per compiere i lavori si calcola come la somma delle durate per i nodi in serie, mentre, si conteggia il valore massimo per i rami in parallelo. Per l'esempio il tempo stimato per la realizzazione è 73 giorni lavorativi, il modello può prevedere anche dei ritardi. Il ritardo di alcune attività non inficia la consegna dei lavori, quindi il tempo totale, mentre in altre fa slittare la durata totale. Ad esempio, il ritardo di un giorno per la realizzazione del compito 11, "intonacatura vano 1" non fa allungare la durata totale, mentre uno stesso ritardo per il punto 20 ritarda il tutto.

ALCUNI ESEMPI

Di seguito sono riportati alcuni esempi per i quali la programmazione di lavori può concretamente migliorare l'efficienza dell'intero sistema.

Esempio 1. Un ufficio postale, dove nel caso più generale vi sono più sportelli che servono più utenti. Si suppone che in base alla operazione da fare ogni utente conosca a priori il tempo di permanenza allo sportello. Inoltre, si ipotizza che un utente possa richiedere il servizio di più sportelli. Infine, si può pensare che vi siano dei servizi a maggiore priorità di altri, per esempio il servizio telegrammi. È un caso in cui sono note le attività e le risorse, rispettivamente i servizi richiesti dai clienti e gli sportelli. Ovviamente, tutti (ufficio e cittadini) hanno interesse affinché si termini il prima possibile nel rispetto delle code e delle priorità.

Esempio 2. La CPU è una risorsa di straordinaria efficienza e velocità, per questo è in grado di soddisfare le esigenze di più programmi, in termini più generici più processi. Uno dei compiti del sistema operativo è trovare delle strategie per soddisfare le richieste dei vari processi di utilizzo della CPU, nella modalità multiprogrammazione. I processi possono avere diverse priorità. Si noti che i metodi più efficienti consentono di interrompere l'esecuzione di un processo per riprenderla poi, tale modalità è conosciuta come preemption. Qui le varie attività attingono ad un'unica risorsa. Si vuole ovviamente minimizzare il tempo di attesa di ogni singolo processo.

Esempio 3. Una piccola azienda produce scatole per dolci di diversa dimensione e con diverse etichette. La catena di montaggio prevede quattro fasi che vanno realizzata in sequenza: taglio del cartone, piegatura, stampa etichette e incollaggio. Ogni fase è associata al lavoro di una macchina specifica. Anche qui l'obiettivo è minimizzare il tempo di produzione, e quindi di non accumulare ritardi rispetto ai tempi previsti.

TERMINOLOGIA

Come accennato, in questi anni si è costruita una significativa teoria su questo argomento. Gli esempi proposti sono solo particolari tipi afferenti alla famiglia dello scheduling, in realtà le situazioni che possono presentarsi sono tante ed è quindi necessario produrre una vera classificazione oltre che standardizzare il lessico da usare. Un modo più articolato per definire i problemi di scheduling indica che: "bisogna svolgere un insieme di attività che attingono a più risorse, spesso scarse, in una quantità di tempo, ottimizzando una funzione obiettivo". Solitamente l'obiettivo è minimizzare il tempo. Nell'ambito della teoria dello scheduling le risorse sono indicate con il termine di provenienza industriale di *macchine*, mentre per le attività è uso comune fare riferimento a *task*. Importante è anche il concetto di *job* che indica più task in relazione tra loro. In altri termini più task (attività) possono costituire un unico job (lavoro), come nel caso dei sistemi operativi dove più task possono costituire un processo. Due variabili sempre note sono il numero di macchine e il numero di job, indicate rispettivamente con m e n. Per il job si distinguono diversi tempi:

Durata pij: tale tempo conosciuto anche come tempo di processamento, indica il tempo richiesto dalla macchina i per eseguire il task per il job j. Si tratta dell'i-esimo task del job j. Le durate sono

OTTIMIZZAZIONE COMBINATORIA

Si tratta di particolari problemi di ottimizzazione che trovano la soluzione nell'applicazione di strumenti del calcolo combinatorio. Si attua dove si hanno degli oggetti o più in generale entità che vanno disposte, combinati, permutate o assegnate tra loro. Il tipico esempio di problema di ottimizzazione combinatoria è knapsack ossia il ladro che avendo a disposizione un sacco di una determinata capienza si trova davanti al dilemma di quali oggetti rubare tenendo conto delle dimensioni e del valore di ogni oggetto, nonché ovviamente della capacità del sacco.

tempi certi, deterministici, e indicano dati sempre richiesti per il problema di scheduling. Come il singolo valore (costituito da due indici) suggerisce tali informazioni possono opportunamente essere rappresentate con una matrice costituita da m righe e n colonne.

Release date rj: è il tempo di rilascio che indica rispetto al tempo iniziale ti=0 quando è possibile cominciare l'esecuzione del job. Spesso esistono dei tempi tecnici che non consentono la disponibilità di risorse per l'avvio di alcune attività. In tal caso un modo naturale per rappresentare queste informazioni è mediante un vettore di n elementi.

Due date dj: è il tempo di consegna. Sempre rispetto al tempo iniziale ti=0 indica entro che tempo bisogna terminare il job. È facile da associare proprio ai tempi di consegna postali, si indica cioè un termine entro cui va evaso l'ordine. Il non rispetto di tali scadenze può comportare penali o rallentamenti dell'intero sistema. Ovviamente, è necessario che in fase di progettazione ci sia una congruità tra i vari tempi descritti, il altre parole la durata di un job calcolata coma la somma delle durate dei singoli task sommata a eventuali tempi di rilascio deve essere minore al più uguale del tempo di consegna. Anche qui un vettore è una giusta struttura da usare.

Tempo di completamento Cj. È il tempo in cui termina il job j o analogamente il tempo che impiega l'ultimo task del job j per terminare. I tempi sono riferiti al tempo iniziale ti=0 di avvio del processo.

Di seguito sono riportati altri tempi che si calcolano dai fondamentali appena proposti. Essi esprimono in modo sintetico diverse esigenze delle scheduler.

Lateness Lj. È la differenza tra il tempo di completamento e il due date ed è riferito al job j. Vale quindi Lj=Cj-dj. Esprime un ritardo se positivo, un anticipo se negativo. Tardiness Tj. Questo indice è uguale alla Lateness nel caso di ritardo, ossia se Lj è positiva, zero altrimenti. Tj=max{0, Lj}.

Oltre ai tempi esiste un altro elemento legato ai job di fondamentale importanza. Si tratta del peso (weight) wj che indica rispetto ad una scala di valori l'importanza di un job. È utile usarlo qualora si imponga una sorta di priorità tra job o anche quando non si ha come unico obiettivo quello di minimizzare dei tempi, ma bensì quando si deve tenere conto di altri elementi del job opportunamente espressi dai pesi.

CLASSIFICAZIONE E SPECIFICHE

La principale classificazione dei problemi di scheduling riguarda il sistema produttivo od organizzativo e l'uso che si fa delle risorse-macchine. A tale proposito si distinguono tre tipi di architettura dei sistemi:

Macchina singola. È questo il caso più semplice per il quale tutti i job richiedono la stessa macchina per essere eseguiti. Nel caso specifico un job non è scomposto in attività elementari, è quindi ci si può indifferente riferire al job o al task. L'esempio più calzante di questa tipologia è proprio il sistema operativo multiprogrammato a singola CPU, dove quest'ultima risorsa che sappiamo essere indivisibile ma interrompibile viene assegnata ai diversi job (processi), per fissate quantità di tempo al fine di servire tutti i processi e di minimizzare il tempo medio di attesa di ogni processo.

Flow shop. Il termine flusso suggerisce il funzionamento del metodo. Le m macchine sono poste in serie e ogni job per essere eseguito sarà servito nell'ordine da tutti i task associati alle m macchine, partendo dalla 1, poi passando alla 2 e così via. L'analogia questa volta richiama i sistemi produttivi a catena di montaggio tipici delle industrie automobilistiche, dove l'auto inizialmente solo struttura, ossia mera scocca, viene passata alle innumerevoli macchine, dove sono presenti operai o anche solo robot, che montano man mano le varie componenti dell'automobile.

Job shop. È questo il caso più generale. Qui non esiste un ordine precostituito di macchine da visitare. Ciascun job può usare anche solo alcune macchine e nell'ordine che necessario. Ogni job può avere un suo ordine. Evidentemente si tratta di un caso più generale rispetto al precedente. L'esempio dell'ufficio postale afferisce a questo tipo di architettura.

Vi sono molti altri elementi che caratterizzano un problema di scheduling, mi limiterò ad indicare i più importanti al fine di avere un quadro d'insieme sufficientemente esaustivo. Una specifica che spesso si trova, come è accaduto nel nostro esempio della casetta, sono dei *vincoli di precedenza*. Ossia come è stato descritto nell'esempio per alcune attività devono essere rispettate delle propedeuticità; delle precedenze tra task, non si può costruire il tetto di una casa se non sono stati fatti i pilastri e i muri. Tali vincoli possono non solo coinvolgere i task di un singolo job ma anche task di diversi job o ancora differenti job. Il modo migliore per descrivere le precedenze è usare un grafo così come è stato proposto nell'esempio. Un'altra specifica è la *preemption*, il cui problema è conosciuto come preemptive. In alcune situazioni è consentito interrompere un job per eseguirne un altro più im-

RIFERIMENTI AD ALTRI ARTICOLI

Nella sezione soluzioni a cura di Fabio Grimaldi, dal numero 89 al numero 95, sono state affrontate le fondamenta teoriche dei sistemi operativi. In particolare nei numeri 89 e 90 si sono osservati gli algoritmi di scheduling, riferiti ovviamente al solo caso del Kernel dei sistemi operativi e quindi ad un campo di applicazione ben preciso, sebbene il più conosciuto.

portante, I tempi di *set-up* uij, sono presenti in alcune realtà industriali ed indicano il tempo che la macchina i richiede per la propria configurazione e avvio (setup) per poter eseguire il job *j*. Esistono macchine che servono a più scopi, per ognuno di essi possono essere richiesti tempi di avvio. Si pensi alle tanto diffuse stampanti multifunzione, a seconda delle operazioni che devono svolgere esistono dei tempi di avvio, ad esempio se si deve fare una scansione bisogna prevedere i tempi di avvio del software per l'acquisizione. Alcune architetture hanno la specifica *blocking* o la più restrittiva no wait. Nel flow shop ogni macchina solitamente dispone di un buffer, può capitare che il buffer della macchina i sia pieno così il job corrente è costretto a fermarsi inattivo sulla macchina precedente *i-1*, favorendo così l'effetto indesiderato di riempire più rapidamente anche questo buffer; rallentando quindi l'intero processo produttivo. Si pensi al caso di una catena di montaggio per la costruzione di auto, se in un punto della catena si verificano problemi (forzata indisponibilità dell'addetto, guasto della macchina, pezzo da montare non idoneo, etc) gli altri job che dopo un po' di tempo arrivano allo stesso punto per essere trattati dovranno attendere in apposite aree di sosta o lungo la catena. Tali spazi sono limitati e se si saturano la macchina precedente non potrà passare il suo lavoro al nodo bloccato e incomincerà a intasare il proprio buffer. Esistono dei tipi di flow shop conosciuti come no-wait per i quali non sono previsti neanche dei buffer sulle singole macchine, cosicché un interruzione di una qualsiasi macchina in breve tempo paralizza l'intero processo.

OBIETTIVI

L'obiettivo è trovare una soluzione ottima ad un problema di programmazione dei lavori. Una qualsiasi soluzione, anche non ottima, è uno schedule. Con questo termine quindi si indica una descrizione completa dell'assegnamento temporale dei task di tutti i job alle macchine che essi richiedono. Nel caso preemptive bisognerà indicare anche le interruzioni specificando per ogni task interrotto i tempi di sospensione e riavvio. Nello schedule si indicano anche i tempi di inizio di ogni task. Qualora tali tempi non siano specificati, ma si indichino solo i task si parla semplicemente di sequenza. Sinteticamente uno schedule si denota con S, mentre S(j) specifica il tempo di inizio del job j per lo schedule S. Le funzioni obiettivo tengono conto delle variabili, dei tempi e delle specifiche che abbiamo sinora descritto.
Makespan Cmax. È il conosciuto massimo tempo di completamento e con riferimento ad uno schedule S indica il massimo tempo di completamento tra gli n job eseguiti. In altri termini è il job che termina per ultimo, che coincide quindi con il completamento di tutti i job. È una misura di tempo rispetto al tempo ti=0 inizio del processo di schedulazione. Sinteticamente si scrive max{C1, C2, …, Cn}.
Max Lateness Lmax. È il valore massimo tra tutti i ritardi degli n job. Se il sistema è particolarmente efficiente potrebbe essere anche un valore positivo cioè un anticipo, in tal caso tra tutti si sceglie il minore anticipo. Sinteticamente max{L1,L2,..,Ln}.
Max Tardness Tmax : max{0, Lmax}.
Somma peseta dei tempi di completamento. È la somma ponderata tra pesi e tempi di completamento. Può

$$\sum_{j=1}^{n} w_j C_j$$

essere espressa come la sommatoria sugli n job.
è il primo caso in cui nella funzione obiettivo compare oltre al tempo un altro parametro che stima altri aspetti, nel caso specifico "l'importanza" dei job. Ovviamente, per tutte queste funzioni l'obiettivo è: minimizzarle. Terminiamo specificando i modi in cui si può esprimere il problema di programmazione dei lavori. Per farlo si possono riportare tre valori. Il primo denota il tipo di scheduling tra i tre elencati: macchina singola, flow shop e job shop indicati rispettivamente con 1, F e J. Il secondo rileva ulteriori caratteristiche del sistema come eventuali vincoli di precedenza, setup, preemption e così via; infine l'ultimo parametro specifica la funzione obiettivo usata. Per i tre esempi fatti nel secondo paragrafo si distinguono tre diverse triplette che distinguono tre diversi problemi di scheduling.

- J, - , $\sum_{j=1}^{n} w_j C_j$
- 1, preemption, $\sum_{j=1}^{n} w_j C_j$
- F, setup, $\sum_{j=1}^{n} w_j T_j$

Il caso della casetta può essere invece descritto come:

- 1, vincoli di precedenza, Cmax

CONCLUSIONI

Lo scheduling comecomprende una ricca famiglia di problemi che possono essere affrontati con metodi consolidati e standardizzati. Nel prossimo appuntamento esamineremo alcuni degli algoritmi più usati oltre che più efficienti tracciando così una percorso metodologico per affrontare queso tipo di problemi.

Fabio Grimaldi